UNIVERSIDAD DE CANTABRIA

Programa de Doctorado de Ciencia y Tecnología

Conjugando Herramientas de Simulación con Aplicaciones y Tecnologías Emergentes en Arquitectura de Computadores

Autor:

Adrián Colaso Diego

Director:

Pablo Abad Fidalgo

Escuela de Doctorado
Santander 2018

Resumen

Nos encontramos actualmente en una época de profundos cambios en el área de Ingeniería de Computadores, desde el hardware subyacente hasta las aplicaciones. El boom del *Big Data* ha propiciado el desarrollo de una notable cantidad de software específico para el almacenamiento y análisis de grandes volúmenes de datos en los últimos años. Sin embargo, la especialización del hardware avanza a un ritmo más desigual, de modo que, parte de este software se ejecuta habitualmente sobre hardware de propósito general. Ante esta situación, se antoja necesaria la precisa caracterización del funcionamiento del hardware existente al ejecutar este tipo de software, con el fin de detectar y dar solución a las potenciales ineficiencias. En lo concerniente a las soluciones hardware a proponer, los límites actuales de escalabilidad que afrontan las tecnologías de fabricación CMOS sugieren la exploración de vías alternativas para continuar diseñando e implementando procesadores que permitan alcanzar mayores cotas de rendimiento. Técnicas en el proceso de fabricación como el *3D-stacking* o tecnologías de memoria no volátiles suscitan desde hace varios años gran interés, tanto desde el ámbito de la investigación como empresarial. La Arquitectura de Computadores, como el nexo entre las aplicaciones y el hardware subyacente, estará a cargo de conjugar de manera eficiente aplicaciones y tecnología. Como herramienta fundamental de esa área, el simulador está obligado a sumarse a dicha evolución para ser capaz de trabajar con las nuevas aplicaciones y las tecnologías emergentes.

Durante el desarrollo de esta tesis se ha trabajado en paralelo para avanzar en los tres puntos mencionados. Utilizando un entorno (*framework*) de simulación de sistema completo, se ha adaptado tanto la herramienta como la metodología de evaluación, para trabajar con aplicaciones asociadas a entornos de computación distribuida (*Cloud Computing*), realizando propuestas arquitecturales basadas en tecnologías de memoria no volátil. El punto de partida del trabajo ha consistido en la evaluación del grado de precisión necesario en la simulación de aplicaciones cuya arquitectura software es cliente-servidor, analizando el error que se comete al simular este tipo de aplicaciones en un único sistema. A la vista de los resultados, se ha hecho necesario trabajar en la modificación de la herramienta de simulación y adaptar la metodología de investigación para adecuarla a los requisitos de las aplicaciones emergentes. Una vez adaptada la metodología, se ha llevado a cabo una exhaustiva caracterización del comportamiento de la jerarquía de memoria presente en los chips al ejecutar aplicaciones emergentes consistentes en varias bases de datos NoSQL (*Not only SQL*) modernas, comparando los resultados con aplicaciones convencionales ampliamente utilizadas en el área. Por último, utilizando la metodología y aplicaciones mencionadas, se ha trabajado con una tecnología de memoria emergente en la realización de una propuesta arquitectural para paliar uno de los puntos débiles de dicha tecnología, la naturaleza variable de la latencia de los accesos. En concreto, se propone la utilización de un mecanismo de identificación de patrones en el acceso a los datos basado en una técnica de *prefetch* hardware para tratar de eliminar la variabilidad de la latencia.

Abstract

Computer engineering area is currently immersed in a period of profound changes, concerning both the underlying hardware and applications. Big Data has favored the development of a significant amount of specific software for the storage and analysis of large data volumes in recent years. However, hardware specialization shows slower evolution, so a part of this software is still executed on general purpose hardware. Under these circumstances, it seems necessary to precisely characterize the existing hardware behavior when executing this kind of software, in order to detect and deal with the potential inefficiencies. As for the hardware solutions to be proposed, the current scalability limits faced by CMOS fabrication technologies suggest the exploration of alternative ways to continue designing and implementing processors to reach higher levels of performance. Novel process fabrication techniques, such as 3D-stacking or non-volatile memory technologies, have been of great interest for academia and industry during the last years. Computer architecture, as the nexus between applications and underlying hardware, will be in charge of efficiently combining these applications and technologies. As the basic tool in this area, the simulation framework must take part in that evolution. This kind of tools should be able to precisely simulate emerging technologies and also to work with novel applications.

In this thesis, we have worked in parallel to make progress in the three aspects previously mentioned. Making use of a simulation framework able to run full system simulations, we have adapted both the simulation tool and the evaluation methodology in order to work with applications that are representative of cloud computing environments, making architectural proposals based on non-volatile memory technologies. This work starts by evaluating the accuracy required to simulate applications with client-server software architecture, analyzing the induced error if the methodology is simplified to single-node simulations. According to the results obtained, it has been necessary to actively work on the simulation tool, adapting the research methodology to the requirements of emerging applications. With that methodology successfully adapted, we have carried out an exhaustive behavior characterization of the on-chip memory hierarchy when executing several modern NoSQL databases, comparing the results with conventional applications widely used in the area. Finally, using both the adapted methodology and emerging applications, we have worked with novel memory technology to make an architectural proposal to mitigate one of the main drawbacks of this technology, the variable nature of access latency. In particular, we propose the use of a mechanism inspired by hardware prefetching techniques, able to identify patterns in memory references to minimize this variability.

Tabla de Contenido

Lista de Figuras

Lista de Tablas

1 Introducción

Actualmente los computadores forman parte de la vida diaria de gran parte de la población del planeta, más incluso de lo que nos damos cuenta. Su presencia no se limita al formato más convencional (equipos de sobremesa, portátiles o servidores), pues diferentes tipos de microprocesadores están presentes en objetos cotidianos, tales como los electrodomésticos o los medios de transporte, desde el coche más sencillo hasta el avión más avanzado. De hecho, los *smartphones* que gran parte de la población utiliza y lleva consigo en todo momento incluyen un microprocesador en su interior, aunque muchos de los usuarios no sean conscientes de la tecnología subyacente de sus dispositivos. Hoy en día es difícil encontrar un campo donde la informática no esté presente en mayor o menor medida. Se utiliza de manera masiva en administración, industria, medicina, educación, etc., siendo un pilar en cualquier campo de relevancia.

Dado el imparable afán de superación y mejora del ser humano, los propios usuarios hemos sido, somos y seremos inagotables demandantes de mayores capacidades de estos dispositivos, deseando que sean capaces de resolver problemas más ambiciosos, en menor cantidad de tiempo. Las aplicaciones evolucionan y son cada vez más complejas (demandando, claro está, más recursos), lo que, sin duda, impulsará a los fabricantes a seguir evolucionando el hardware en el futuro. Un ejemplo claro de dicha evolución es el campo de la medicina moderna, donde los computadores permiten realizar, hoy en día, tareas sumamente complejas, inimaginables hace tan solo unas décadas. Pruebas diagnósticas, como la tomografía axial computarizada o las resonancias magnéticas, que permiten ver el interior del cuerpo humano con mucha precisión, serían imposibles sin computadores [1] para combinar las distintas medidas que se llevan a cabo durante la exploración, generando resultados interpretables por los médicos. Por otra parte, la utilización de robots en las cirugías está cada vez más extendida, permitiendo incluso a los especialistas llevar a cabo las operaciones sin estar en la misma sala que el paciente [2]. El futuro plantea retos tan ambiciosos como la búsqueda de tratamientos eficaces contra enfermedades como el cáncer o el alzhéimer, cuyos avances serán soportados, en gran medida, por la capacidad de cálculo disponible [3]. Ejemplos claros de esta dependencia, como la colaboración entre IBM y el instituto de investigación y tratamiento contra el cáncer *Memorial Sloan Kettering* en materia de inteligencia artificial [4], muestran la necesidad de seguir evolucionando los computadores actuales.

En gran medida, la adopción del computador como herramienta de trabajo en todos los campos mencionados ha sido posible gracias a la rápida evolución del hardware subyacente que, con gran esfuerzo, lo ha transformado radicalmente en solo 50 años. Hoy en día, un simple *smartphone* tiene más capacidad de cómputo que un *mainframe* de hace décadas y su tamaño es tan reducido que cabe en un bolsillo. Este ritmo de evolución ha estado fuertemente marcado por la conocida "Ley de Moore" [5], que en 1965 predijo que el número de transistores en un procesador se doblaría aproximadamente cada dos años. La industria, tomando dicha ley como un objetivo de mejora de rendimiento, ha conseguido que se cumpla hasta el día de hoy, alcanzando 10nm en el proceso de fabricación y estimando densidades aún mayores en un futuro cercano [6]. En este proceso evolutivo, la tecnología no ha sido el único elemento facilitador del progreso en el rendimiento de los procesadores, jugando la Arquitectura de Computadores un papel igual de importante [7]. Desde el primer microprocesador comercializado por Intel a finales del año 1971 [8], las mejoras que paulatinamente han ido modificando la arquitectura son responsables de una parte importante del progresivo aumento del rendimiento de los procesadores. Tal evolución se

plasma en elementos actuales de la arquitectura tales como los *pipelines* segmentados y superescalares, los predictores de saltos, las jerarquías de memoria *cache* multinivel, el *Simultaneous Multithreading* (SMT), la ejecución fuera de orden, la ejecución especulativa de instrucciones en memoria, etc. La micro-arquitectura de un procesador actual resulta extremadamente compleja al compararla con las de los primeros procesadores. Un análisis detallado de las diferencias tecnológicas y arquitecturales, entre el primer procesador de Intel y uno actual, permite comprobar los increíbles avances realizados. La evolución tecnológica ha permitido avanzar de un procesador (Intel 4004 [8]) con 2300 transistores, tecnología de fabricación de 10 micrómetros y frecuencia de reloj por debajo del MHz a uno (Intel *Kaby Lake* [9]) con más de 1.000 millones de transistores ($x10^7$), tecnología de fabricación de 14nm ($x10^3$) y frecuencia de operación entre 3 y 4 GHz ($x10^4$). Las diferencias son igual de llamativas atendiendo a aspectos meramente arquitecturales (ver Figura 1). Así, el 4004 presenta una arquitectura muy sencilla con una única unidad aritmético-lógica (una estructura hardware capaz de realizar distintas operaciones aritméticas y lógicas), unos pocos registros y la lógica de control necesaria, mientras que elementos arquitecturales básicos hoy en día como la segmentación del *pipeline* o la memoria *on-chip* no estaban presentes. En contraste, la arquitectura *Kaby Lake* posee una complejidad mucho más elevada, implementando multitud de mejoras arquitecturales como las mencionadas previamente (predicción de saltos, las *caches on-chip*, la ejecución fuera de orden, etc.). Así, mientras que el 4004 era capaz de ejecutar como máximo varias decenas de miles de instrucciones por segundo, los procesadores actuales están compuestos por varios *cores*, cada uno capaz de ejecutar miles de millones de instrucciones por segundo.

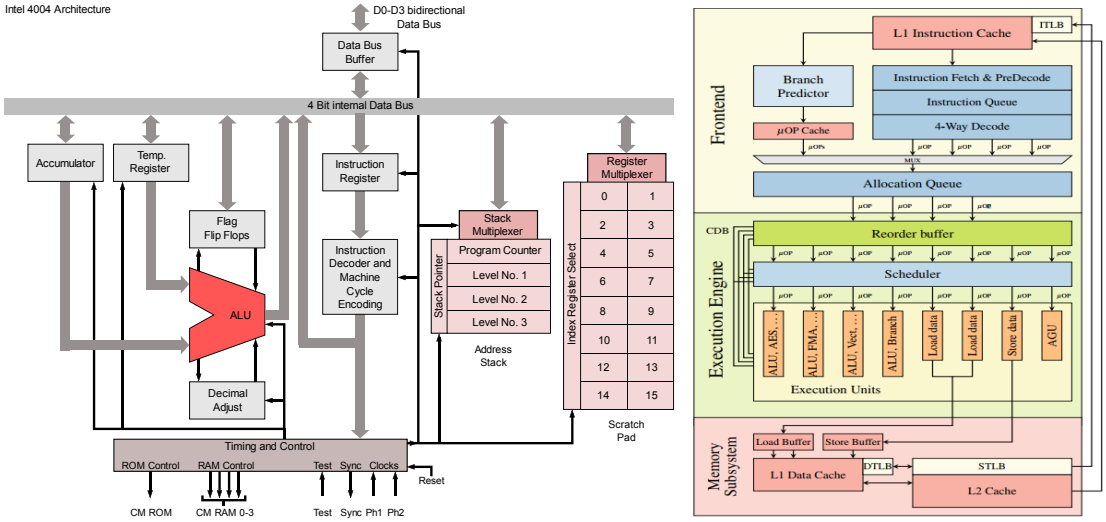

Figura 1 – (Izquierda) Arquitectura del procesador Intel 4004. (Derecha) Arquitectura Intel Skylake.

Desafortunadamente, la inercia en el desarrollo del substrato tecnológico, que ha facilitado la evolución del procesador, muestra síntomas de agotamiento, las predicciones indican que la industria no continuará en la dirección actual durante mucho más tiempo [10]. La miniaturización de los transistores presenta cada vez más problemas, lo que implica un incremento de costes difícilmente asumible [11]. El rediseño de la geometría de los transistores incorporando la tercera dimensión [12] ha permitido llegar a procesos de fabricación en 10nm [13], estimando poder alcanzar los 5nm en los próximos años, pero a partir de ese punto no parece estar claro cómo continuar [14]. Esto no necesariamente significa la muerte de la "Ley de Moore" como meta de rendimiento, puesto que lo único que está en tela de juicio actualmente es la continuidad de los procesos de fabricación basados en tecnología CMOS *(Complementary Metal-Oxide-*

Semiconductor) [15]. El uso de nuevos materiales o técnicas como el *3D-Stacking* permitirán con seguridad continuar transformando en rendimiento la creciente cuenta de transistores en el chip.

Los grandes retos a los que se enfrentan los computadores en un futuro cercano no se limitan a los problemas de agotamiento del proceso de fabricación CMOS o la aparición de múltiples alternativas tecnológicas. A dichos desafíos se ha unido actualmente la vertiginosa evolución del software al calor de fenómenos como el *Big Data* o el *Internet of Things* (IoT), que obviamente han llegado como consecuencia de la tremenda potencialidad del desarrollo del computador. Esto nos sitúa en un momento de grandes cambios hardware y software de manera simultánea, jugando de nuevo la Arquitectura de Computadores un papel fundamental para armonizar dicha evolución. A lo largo de este capítulo introductorio, se describirán los retos mencionados, así como posibles caminos en el I+D en el área para hacer frente a dichos retos.

1.1 El Futuro de las Aplicaciones (El Boom del *Big Data*)

La evolución tecnológica ha supuesto un incremento exponencial tanto en la generación como en el almacenamiento de datos para su posterior procesado. La gestión de estos ingentes volúmenes de información, conocida como *Big Data Analytics*, está atrayendo, en los últimos años, mucha atención desde muy diversos ámbitos. Dada la gran disponibilidad de datos existente, hoy en día es posible aplicar complejas técnicas de análisis para la extracción de valiosa información. Amazon es un ejemplo de cómo beneficiarse del *Big Data Analytics* ya que, mediante su uso, logra incrementar las ventas a través de recomendaciones personalizadas de productos a los usuarios o de la optimización de precios, y desarrollar estrategias para la optimización de la logística consiguiendo ahorrar costes y reducir los tiempos de entrega [16]. El *Big Data* se presenta como una gran oportunidad para la obtención de información, pero al mismo tiempo supone enormes desafíos. Debido a que el análisis de los grandes conjuntos de datos excede las capacidades, tanto del software como del hardware, ambos se han visto obligados a evolucionar en los últimos años.

Dada su elevada complejidad, el proceso de gestión y análisis se estructura habitualmente en varias etapas, utilizando cada una de ellas software específico para sus competencias. Dicho proceso comienza con la adquisición de los datos (sensores, ficheros de *log*, etc.), siendo éste un proceso crítico, dado el actual ritmo de generación de datos, que puede superar, ampliamente, las capacidades de almacenamiento convencionales, forzando en muchas ocasiones a realizar labores de pre-procesado (eliminación de repeticiones o de datos incompletos) para optimizar las etapas posteriores. Por la misma razón, los servicios de almacenamiento necesitan dar soporte a infraestructuras que permitan gestionar el gran volumen de datos, así como proporcionar interfaces eficientes para las posteriores tareas de análisis. El análisis de los datos pre-procesados y almacenados es la etapa capaz de dotar de valor potencial a dicha información, a través de la posibilidad de extracción de información valiosa oculta en complejas relaciones de los datos.

En la era del *Internet of Things*, el elevado ritmo de generación de datos y los, cada vez más estrictos, requisitos de inmediatez (*real-time analytics*) sobrepasan las capacidades actuales de almacenamiento y cálculo, forzando a gran parte del software, surgido a la sombra del *Big Data*, a hacer uso de entornos de computación basados en la nube. El *Cloud Computing* se alza como el entorno más apropiado para el *Big Data*, logrando proveer los recursos de almacenamiento y cálculo necesarios en base a hardware de propósito general, organizado en grandes *clusters* distribuidos geográficamente. Así, gran parte de las aplicaciones desarrolladas para este tipo de entornos (*Big Data*) se sustentan en la ejecución distribuida (escalado horizontal), forzadas por el volumen de datos en particular con el que deben trabajar. De hecho, la estrecha relación existente

entre el *Big Data* y el *Cloud Computing* ha impulsado el rápido desarrollo de ambas. Sistemas de ficheros distribuidos como HDFS [17] o QFS [18], modelos de programación distribuidos como MapReduce [19] o Spark [20], bases de datos NoSQL [21] como Cassandra [22] o MongoDB [23] y herramientas de *Machine Learning* y *Deep Learning* como Tensorflow [24] y PyTorch [25] son solo algunos ejemplos de aplicaciones software con menos de 10 años de antigüedad y actualmente presentes en gran parte de los entornos de *Cloud Computing* existentes.

Es evidente que, tanto el software como las infraestructuras de cálculo, se han especializado a los requerimientos del *Big Data*. Sin embargo, durante estos años de evolución, los procesadores han sufrido diferentes niveles de especialización. Mientras que áreas como el *Machine Learning* han visto cómo el desarrollo software era acompañado por infraestructuras hardware de propósito específico [26], los procesadores utilizados en cualquier infraestructura de cálculo distribuido para trabajar con herramientas de almacenamiento masivo de datos son de propósito general y no han sido diseñados específicamente para estas aplicaciones. Así, se han constatado importantes ineficiencias [27][28] que degradan gravemente el rendimiento en la ejecución de este tipo de aplicaciones en los entornos mencionados. Parece evidente la necesidad de avanzar en la exhaustiva caracterización de este tipo de aplicaciones para poder diseñar arquitecturas más acordes a los requerimientos observados. Bien a través del rediseño o incorporando nuevas tecnologías, se puede optimizar el hardware con el fin de mejorar tanto el rendimiento como la eficiencia al ejecutar aplicaciones *Big Data*. Es, por ello, fundamental que las herramientas de simulación sean capaces de adaptarse para la evaluación de este tipo de aplicaciones.

Una parte del trabajo realizado en esta tesis ha consistido en la realización de una detallada caracterización del comportamiento de aplicaciones emergentes en la jerarquía de memoria, utilizando una metodología basada en simulación [29]. La naturaleza flexible de esta metodología ha permitido profundizar más allá de donde han llegado trabajos previos de caracterización basados en la obtención de estadísticas mediante la utilización de contadores hardware [27][28] [30][31][32], una metodología rígida que no permite la realización de importantes experimentos.

1.2 El Substrato Tecnológico

Sin duda, el fuerte empuje del software se verá apoyado por la evolución tecnológica en su búsqueda de mayores cotas de rendimiento. Dado el rápido avance tecnológico, hacer predicciones más allá de 5-10 años se torna extremadamente especulativo, pero actualmente en posible intuir algunos de los cambios tecnológicos que veremos en un futuro cercano. Las tecnologías de fabricación actuales todavía tienen recorrido gracias a mejoras en el proceso de fabricación (Transistores 3D [12] y apilado vertical [33]), y a la aparición de tecnologías emergentes o modelos de computación alternativos (aceleración hardware basada en GPU (*Graphics Processing Unit*) [34] y TPU (*Tensor Processing Unit*) [26]) parecen ser un camino razonable para continuar incrementando las prestaciones de las futuras generaciones de procesadores.

La continua miniaturización del tamaño del transistor en el proceso de fabricación CMOS es finita. Sin embargo, gracias a la gran cantidad de avances en este proceso, la “Ley de Moore” parece resistirse a desaparecer. La utilización de nuevos materiales y el rediseño de la geometría del transistor parece abrir nuevos caminos para seguir avanzando. Así, la utilización de nuevos materiales (mezcla silicio-germanio [35] o de elementos de los grupos III-V de la tabla periódica [36]) o la propuesta de geometrías de transistor alternativas (*Gate-All-Around FET* [37]) parecen permitir continuar miniaturizando los transistores. Adicionalmente, se trabaja en alternativas

consistentes en la incorporación de la tercera dimensión al proceso de fabricación, permitiendo apilar y conectar verticalmente capas de silicio (*3D Stacking* [33]). A través de pilares verticales que atraviesan todas las capas (denominados TSVs o *Through-Silicon Vías* [38]) se logra incrementar la densidad de transistores por mm^2 sin tener que recurrir a procesos de miniaturización. A pesar de presentar retos para su utilización en la fabricación de procesadores [39], se trata de una tecnología con suficiente madurez como para que fabricantes como Samsung, Fujitsu o AMD presenten productos comerciales que incluyen el apilado vertical en su tecnología de fabricación, apilando hasta 8 capas de memoria DRAM (*Dynamic Random Access Memory*)[40].

Al margen de las mejoras en el proceso de fabricación, múltiples tecnologías de memoria no volátil emergen como candidatas potenciales para la sustitución de la tecnología SRAM (*Static Random Access Memory*), en la cual están basadas las actuales jerarquías de memoria *on-chip*. Los síntomas de agotamiento en la miniaturización CMOS, así como problemas graves tales como el consumo de potencia [41], han disparado el interés en la búsqueda de una tecnología capaz de reemplazar a la actual. Las tecnologías propuestas se basan en diferentes fenómenos físicos para su funcionamiento, si bien comparten una serie de características, como una mayor densidad de integración debido al menor tamaño de sus celdas y un menor consumo de potencia estática. Entre la multitud de propuestas disponibles en la literatura [42], tecnologías como *Conductive Bridge RAM* (CBRAM) [43] y *Spin- torque Transfer RAM* (STT-RAM) [44] parecen contar con un mayor nivel de madurez. Ambas cuentan con productos comerciales, enfocados a la fabricación de unidades de disco (como *cache* de escritura) [45] o memoria principal [46][47]. El creciente interés en este tipo de tecnologías alternativas implica la rápida aparición de nuevas propuestas que intentan competir con las mencionadas. Tal es el caso de las *Racetrack memories* [48], una evolución de la tecnología STT-RAM que permite el almacenamiento de múltiples bits en una sola celda.

Por el momento, la completa sustitución de la tecnología SRAM (hasta los niveles más bajos de la jerarquía de *cache on-chip*) no parece cercana. Estas tecnologías presentan desafíos con respecto a la tecnología SRAM [49], como son los tiempos y la energía de los accesos o el *endurance* (el número máximo de ciclos de escritura). El trabajo de los arquitectos de computadores jugará un papel fundamental para la adopción de estas tecnologías, desarrollando mecanismos capaces de paliar sus actuales carencias. Es, por tanto, necesario, implementar estas tecnologías tan prometedoras en las herramientas de trabajo para evaluarlas de manera apropiada.

El trabajo final de la presente tesis se ha centrado en uno de los principales desafíos que conllevan las tecnologías de memoria emergente denominadas *Racetrack Memories* [48], mediante la proposición y evaluación de una solución micro-arquitectural para tratar de acercar la integración de esta tecnología dentro del chip [50]. En concreto, la naturaleza variable de los accesos supone un condicionante para su integración en los niveles de la jerarquía más cercanos al procesador, de modo que, se propone un mecanismo basado en una técnica de *prefetch* hardware para la identificación de patrones en los accesos que permita reducir la parte variable de la latencia de un modo más afectivo que los mecanismos que explotan la localidad existentes en la literatura [51] [52].

1.3 El Futuro en el I+D en AC

La Arquitectura de Computadores ha sido el claro nexo entre el substrato tecnológico y el software, a cargo de convertir esa densidad creciente de transistores en suficiente rendimiento para aplicaciones cada vez más demandantes. Sin duda, seguirá jugando un papel similar en un

futuro cercano. Estando en un momento de cambios profundos, tanto en la parte software como en el hardware, parece evidente que la metodología de trabajo del área debe evolucionar en la misma medida. Los *benchmarks* deben incorporar cargas de trabajo representativas de las herramientas software surgidas en los últimos años, mientras que las herramientas de simulación deben ser capaces de emular, tanto las tecnologías emergentes como los nuevos entornos de ejecución (*cloud*), de manera ajustada a la realidad.

1.3.1 *Benchmarking* y Nuevas Aplicaciones

La cantidad y variedad de software en uso, sobre cualquier plataforma incluyendo un procesador de propósito general, es infinita, forzando a que, en la práctica, la evaluación se deba llevar a cabo ejecutando conjuntos (*suites*) de aplicaciones representativas denominadas *benchmarks*, de modo que, mediante la ejecución de unas pocas aplicaciones, se intenta caracterizar un comportamiento lo más genérico posible. Gran parte de las propuestas realizadas en el área de Arquitectura de Computadores hacen uso de este tipo de herramientas, como demuestra la exploración de los trabajos publicados en los tres congresos más importantes del área de Arquitectura de Computadores (ISCA, MICRO y HPCA) de los últimos años. En la mayoría de los casos, las propuestas se limitan al uso de *benchmarks* tradicionales como SPEC [53] y PARSEC [54] (ver Figura 2). Dichos *benchmarks* contienen aplicaciones representativas, en su mayor parte de software ejecutado en entornos de escritorio, tales como un compilador (*gcc*), un compresor (*bzip2*) o un compresor de vídeo (*h264ref*). Desafortunadamente, este tipo de aplicaciones son poco o nada representativas de aquellos entornos de ejecución surgidos a partir del *Big Data*. La relevancia que han cobrado este tipo de entornos emergentes, así como la ausencia de software de evaluación adecuado, han forzado la aparición en los últimos años de multitud de *benchmarks* relacionados con el *Big Data*. Sin embargo, su adopción en el área se ha visto ralentizada (Figura 2 (categorías denominadas “Emergentes” y “Redes neuronales”)) debido, en gran parte, a la complejidad de este tipo de aplicaciones. Herramientas de trabajo centradas en el análisis preciso de la micro-arquitectura del procesador tienen dificultades para adaptarse a entornos distribuidos (la arquitectura de las aplicaciones emergentes es cliente-servidor, donde la parte del servidor puede estar, a su vez, compuesta por más de una instancia, y requiere simular tantos sistemas como sean necesarios) y a aplicaciones con un *stack* software significativamente más complejo (múltiples capas de *middleware*). Es, por tanto, necesario, hacer un esfuerzo por adaptar los entornos de simulación para que sean capaces de trabajar con este tipo de aplicaciones.

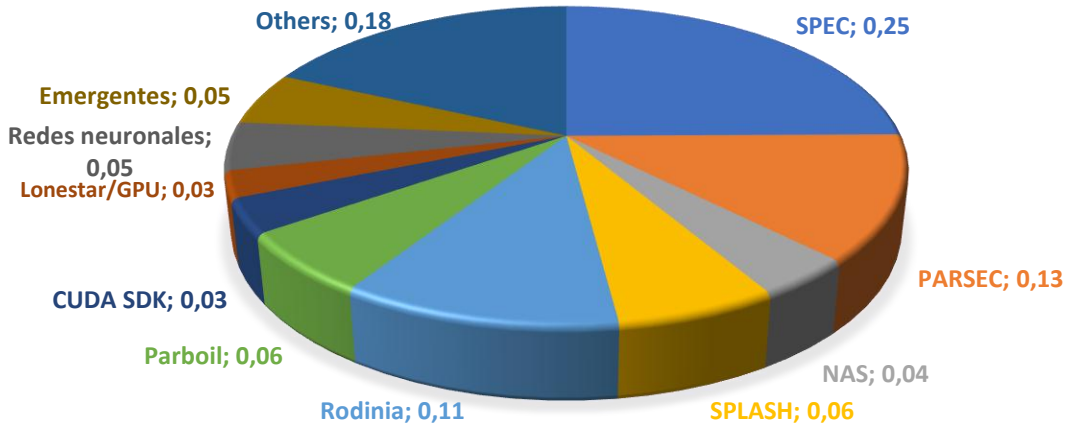

Figura 2 – Exploración de los benchmarks utilizados en el área de Arquitectura de Computadores.

1.3.2 El Simulador como Herramienta de Trabajo

La simulación es la metodología de investigación más utilizada en el área de Arquitectura de Computadores, ofreciendo un punto medio en términos de costes y precisión de los resultados. Evita los elevados costes económicos y de tiempo del prototipado hardware y la imprecisión de los modelos analíticos [55]. Repitiendo la exploración previa sobre los trabajos publicados en ISCA, MICRO y HPCA, se confirma la relevancia de esta metodología, siendo utilizada en prácticamente el 70% de los trabajos publicados, tal y como muestra la Figura 3.

En este tipo de metodologías existe un *trade-off* persistente entre la precisión en la simulación y el esfuerzo computacional. Cuanto mayor es el nivel de detalle con el que se modela el hardware, mayor es el tiempo de simulación y viceversa. Independientemente del nivel de detalle, la simulación es una tarea lenta, dado que se trata de un modelo software del hardware subyacente y su velocidad dista mucho de la obtenida con hardware real. Un *equilibrio* similar se revela de nuevo entre la precisión y la "curva de aprendizaje". Un gran nivel de detalle implica un alto esfuerzo de aprendizaje debido a la complejidad de este tipo de herramientas (cientos de miles de líneas de código, mezcla de múltiples lenguajes de programación). Los requerimientos impuestos por el software surgido a raíz del *Big Data*, tales como el notable incremento del *stack* software o el modelo multi-nodo (cliente-servidor), imponen nuevos condicionantes a las herramientas de simulación que pueden influir de manera significativa en los *trade-offs* anteriormente mencionados. Se detallan a continuación los más relevantes.

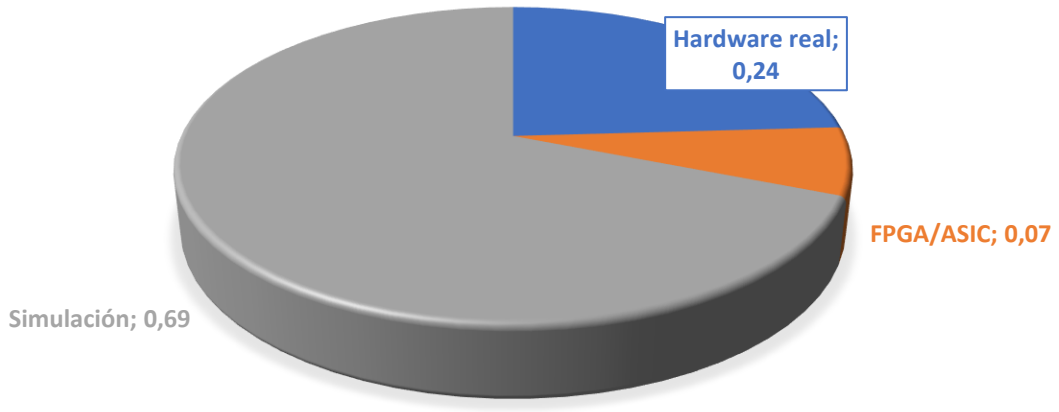

Figura 3 – Exploración de las metodologías de investigación en el área de Arquitectura de Computadores.

En primer lugar, el sistema operativo se vuelve una parte importante de la simulación dado que la cantidad código ejecutado por las nuevas aplicaciones, en modo supervisor, crece notablemente en comparación con las aplicaciones tradicionales [30]. Así, la adopción de las nuevas aplicaciones impone la inclusión del sistema operativo como parte del software. Esto tiene una implicación muy relevante sobre la herramienta de simulación empleada, requiriendo un nivel de detalle capaz de permitir la ejecución de un sistema operativo sin modificar. En segundo lugar, la arquitectura software de las nuevas aplicaciones es distribuida, basada en el modelo cliente-servidor. La correcta simulación de aplicaciones distribuidas impone a los simuladores la capacidad de llevar a cabo simulaciones multi-nodo, desplegando múltiples sistemas y al mismo tiempo comunicándolos entre sí. Finalmente, la estructura de datos necesaria para la ejecución de algunas aplicaciones emergentes supone, en algunos casos, un enorme desafío. Así, el proceso de poblar una base de datos a través de un sistema simulado resulta prácticamente inabordable, en tiempo, dado que es un proceso largo, incluso utilizando hardware real. Simular los miles de millones de instrucciones necesarios para llevarlo a cabo puede requerir incluso años en función

del tamaño de la base de datos que se desee, motivo por el que la aceleración hardware resulta indispensable.

La creciente cuota de mercado de las aplicaciones emergentes y los serios requerimientos que imponen a las herramientas de simulación han motivado una de las principales aportaciones de esta tesis. Concretamente, se ha trabajado en la adopción de un *benchmark* conteniendo aplicaciones emergentes a la metodología de evaluación, siendo necesario realizar las pertinentes modificaciones a la herramienta. Este trabajo ha permitido el uso de aplicaciones emergentes en los capítulos finales de la presente tesis.

Hasta el momento, muy pocos han sido los trabajos capaces de conjugar la utilización de herramientas de simulación detalladas para llevar a cabo evaluaciones en entornos *Big Data*. La disponibilidad de una herramienta que modele, con gran nivel de detalle, el hardware y que, al mismo tiempo, reúna los requisitos descritos resulta complicado. Dicha herramienta ha sido el pilar fundamental sobre el que sustentar el resto de las contribuciones realizadas, detalladas en la siguiente sección.

1.4 Contribuciones de la tesis

El trabajo desarrollado durante esta tesis se ha centrado en la adaptación de herramientas de simulación que permitan realizar propuestas arquitecturales basadas en tecnologías de memoria resistiva, evaluando dichas propuestas a través de *benchmarks* basados en aplicaciones emergentes. Como herramienta básica, se ha empleado *gem5* [56], un simulador de sistema completo ampliamente utilizado en el área de Arquitectura de Computadores. A través de dicha herramienta, ha sido posible llevar a cabo una exhaustiva caracterización de aplicaciones habituales en entornos *Big Data*, como son las bases de datos NoSQL, así como realizar propuestas arquitecturales basadas en tecnologías emergentes de memoria no volátil.

A modo de resumen, las principales contribuciones de esta tesis son:

- Una adaptación de las metodologías de evaluación basadas tanto en *profiling* hardware como en simulación de sistema completo para dar respuesta a los requerimientos de aplicaciones emergentes, concretamente bases de datos NoSQL [57].
- Una detallada caracterización del comportamiento de la jerarquía de memoria *on-chip* para múltiples bases de datos NoSQL, además de la comparación con *benchmarks* tradicionales utilizados en el área. Los resultados muestran conclusiones relevantes, tales como el similar comportamiento, en jerarquía de *cache*, a aplicaciones más convencionales o una significativa uniformidad en comportamiento, independientemente de aspectos como el *workload* utilizado, el tamaño de la base de datos empleado o incluso la base de datos utilizada [29].
- Una propuesta arquitectural para acercar el de uso de una tecnología emergente de memoria no volátil denominada *Racetrack Memory* (RM) [48] en la jerarquía de memoria *on-chip* como sustituta parcial de la tecnología SRAM. El trabajo se ha centrado en solucionar el principal desafío de esta tecnología, la latencia variable de acceso, desarrollando una nueva política de reposicionamiento de la cabeza de lectura y escritura basada en mecanismos de *prefetch hardware* [50].

Todas las contribuciones han sido publicadas o están en proceso de revisión en foros (tanto revistas como congresos) de gran relevancia a nivel internacional en el área de Arquitectura de

Computadores. Cabe destacar, como resultado adicional de la presente tesis, que, gracias al trabajo desarrollado en la herramienta de simulación utilizada por todo el grupo de investigación, los miembros de dicho grupo continúan aprovechando todo este esfuerzo para completar y evaluar sus propios trabajos de investigación. A continuación, se enumeran, en orden cronológico inverso, las publicaciones directamente relacionadas con esta tesis, así como los trabajos de investigación en los que ha colaborado el autor.

1.4.1 Publicaciones en Revistas y Congresos Internacionales

- A. Colaso, P. Prieto, P. Abad, V. Puente, J.A. Gregorio, "Architecting RM preshift through pattern-based prediction mechanisms", Submitted to International Parallel and Distributed Processing Symposium (Notification, January 2019). Preprint temporalmente disponible en: https://www.ce.unican.es/ThesisAdrianColaso/RM-Preshift-Colaso.pdf
- A. Colaso, P. Prieto, J. A. Herrero, P. Abad, "Accuracy vs. Computational Cost Tradeoff in Distributed Computer System Simulation" Submitted to International Symposium on Performance Analysis of Systems and Software (Notification, January 2019). Preprint temporalmente disponible en: https://www.ce.unican.es/ThesisAdrianColaso/Gem5-Multinode-Colaso.pdf
- A. Colaso, P. Prieto, J.A. Herrero, P. Abad, L.G. Menezo, V. Puente, J.A. Gregorio, "Memory Hierarchy Characterization of NoSQL Applications through Full-System Simulation", IEEE Transactions on Parallel and Distributed Systems 29 (5), 1161-1173.

1.4.2 Publicaciones en Congresos Nacionales

- A. Colaso, P. Prieto, J. A. Herrero, P. Abad, V. Puente, J. A. Gregorio, "Precisión vs. Coste Computacional en la Simulación de Sistemas Distribuidos" Actas de las Jornadas Sarteco 2018, Teruel, España.

1.4.3 Participación en otras publicaciones

Además de los trabajos directamente relacionados con el contenido de la tesis, el autor también ha participado en otras publicaciones del grupo de investigación que guardan cierta relación indirecta con el contenido de la tesis.

- P. Abad, P. Prieto, L.G. Menezo, A. Colaso, V. Puente, J.A. Gregorio, "Interaction of NoC design and Coherence Protocol in 3D-stacked CMPs", 2013 Euromicro Conference on Digital System Design, pp. 48-55.
- P. Abad, P. Prieto, L. G. Menezo, A. Colaso, V. Puente, J.A. Gregorio, "Topaz: and open-source interconnection network simulator for chip multiprocessors and supercomputers", sixth IEEE/ACM International Symposium on Networks on Chip (NOCS), pp. 99-106.
- L.G. Menezo, A. Colaso, V. Puente, J.A. Gregorio, "Beneficios del uso de la Red de Interconexión en la Aceleración de la Coherencia", XXII Jornadas de Paralelismo, La Laguna (Spain), September 2011.
- L.G. Menezo, A. Colaso, V. Puente, J.A. Gregorio, "Exploring Coherence Protocol Acceleration through the Interconnection Network", Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems (ACACES), Italy, July 2011.

1.5 Contenido de la tesis

La organización de los restantes capítulos de la tesis es la siguiente:

- El capítulo 2 analiza la revolución del *Big Data*, definiendo en qué consiste y analizando los cambios que ha provocado en el hardware y en el software. Además, se describe con detalle parte de ese software novedoso que ha surgido como consecuencia del *Big Data* y que se va a utilizar en la tesis, las bases de datos NoSQL.
- El capítulo 3 describe en detalle las metodologías de investigación a utilizar en la tesis: el *profiling* a través de los contadores hardware y la simulación de sistema completo, y su adaptación a los requerimientos de las bases de datos emergentes. Se evalúa adicionalmente la problemática de metodologías de simulación no adecuadas.
- En el capítulo 4 se caracteriza el comportamiento en memoria de un conjunto de bases de datos NoSQL utilizando la metodología descrita en el capítulo previo, comparando además los resultados obtenidos para este tipo de aplicaciones con los de los *benchmarks* convencionales utilizados en el área. Se comienza analizando aspectos básicos como el tamaño del *working set* de instrucciones y datos, centrándose posteriormente el análisis en aspectos más complejos como el funcionamiento de los algoritmos de reemplazo o de mecanismos de *prefetching*.
- El capítulo 5 se centra en la adecuación de tecnologías emergentes para entornos tanto convencionales como asociados al *Big Data*. En concreto, se propone el uso de una tecnología de memoria no volátil, las *Racetrack Memories* [51], como sustituta parcial de la tecnología SRAM en la jerarquía de memoria *on-chip*.
- Por último, el capítulo 6 resume las principales conclusiones extraídas del presente trabajo y plantea futuras líneas de investigación.

2 Big Data

2.1 Introducción

El volumen de datos a nivel global aumenta de manera imparable, incrementando el ritmo de generación de nuevos datos a un nivel exponencial (ver Figura 4). La superación en el año 2010 de la barrera del *zettabyte (ZB)* (10^{21} bytes) marcó un punto de no retorno, alcanzando el año siguiente 1,8ZB y según algunas estimaciones pudiendo llegar a los 40ZB en el año 2020 [58]. Si estas previsiones se cumplen, el volumen de datos global se habrá multiplicado por 300 en únicamente 15 años. Entre las múltiples fuentes que están contribuyendo al actual ritmo de generación de datos, caben destacar algunas como las redes sociales, el "Internet de las cosas" (IoT o *Internet of Things*) o la "Industria 4.0".

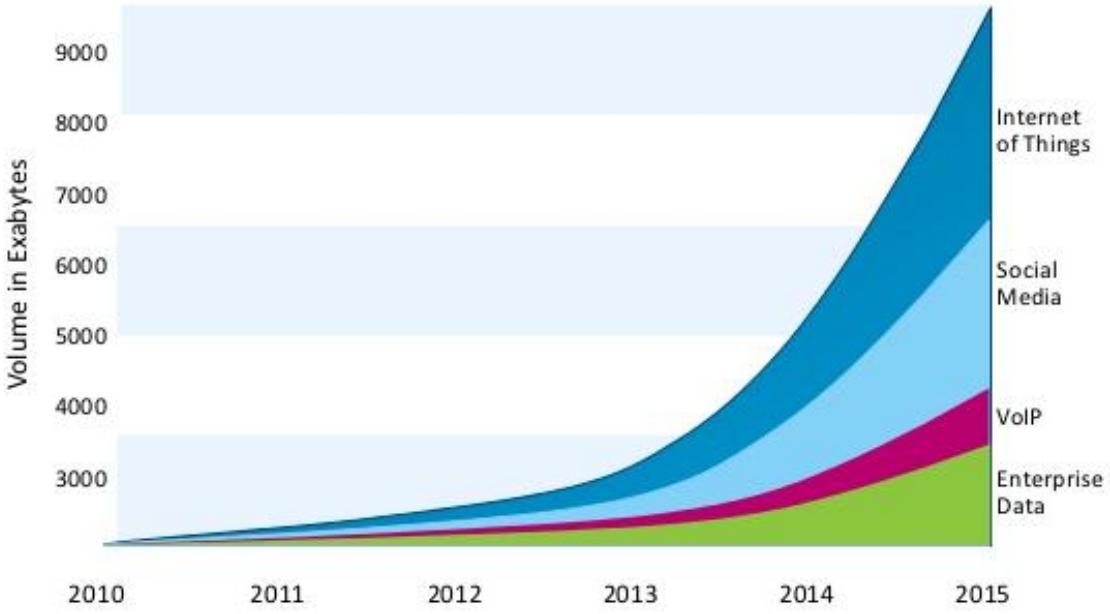

Figura 4 – Evolución y estimación de crecimiento del volumen de datos global [59].

El creciente número de personas que hacen un uso recreativo de internet [60] es uno de los principales contribuyentes al volumen de datos actual. La utilización cotidiana de los distintos motores de búsqueda (se realizan varios millones de búsquedas en Google cada minuto [61]) y la generación de contenido multimedia en redes sociales (cada minuto se generan cientos de horas de vídeos en Youtube [62] o se suben más de 100.000 fotos a la red social Facebook [63]) son dos ejemplos claros de datos en constante generación. Adicionalmente, la penetración actual del uso de dispositivos de reducido tamaño como los *smartphones* o *wearables,* sin limitaciones de portabilidad o restricciones en la conectividad (4G o Wi-Fi), fomenta la continua creación de nuevos datos. Finalmente, el volumen de datos creado en las redes sociales se ve incrementado de forma indirecta, por la recolección de estadísticas de uso de aplicaciones o servicios que llevan a cabo las empresas propietarias. De hecho, en numerosas ocasiones, la información que se crea sobre los usuarios (el usuario como producto) es mucho mayor que la información creada directamente por éstos [58].

La utilización de múltiples sensores (localización, temperatura, humedad, etc.) y actuadores (interacción con el espacio físico) en ámbitos tan dispares como el consumo, la empresa y las

infraestructuras, están a la orden del día. Este tipo de dispositivos, conectados a la red de manera permanente y conocidos como IoT, son también partícipes del ingente volumen de datos actual. La cantidad estos dispositivos no deja de aumentar, previéndose que su número pueda alcanzar los 30.000 millones en los próximos años, según las estimaciones más recientes (ver Figura 5) [64]. La medición de propiedades físicas por parte de cualquier tipo de sensor es capaz de generar información a un ritmo difícilmente alcanzable por el ser humano. Es, por tanto, previsible, que en el futuro la aportación del IoT al crecimiento de datos será aún más significativa de lo que ya supone actualmente.

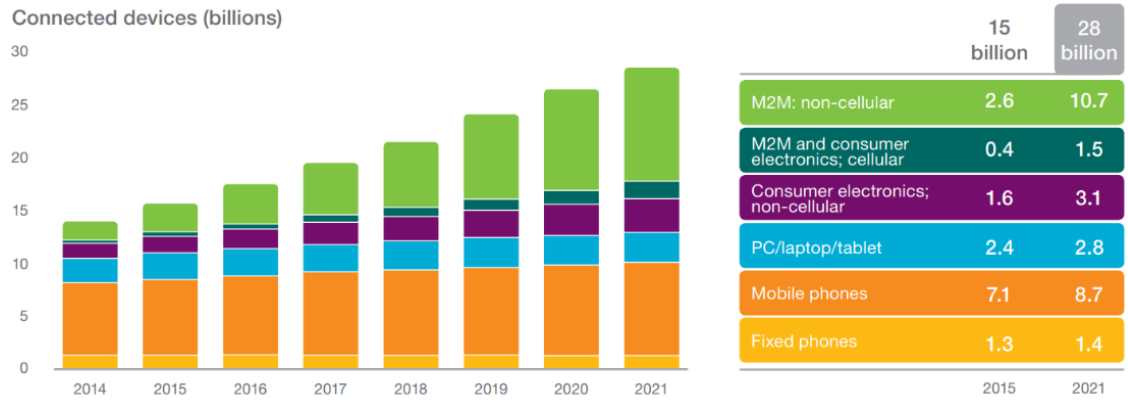

Figura 5 – Predicción del número de dispositivos IoT [65].

Finalmente, el ámbito empresarial no ha sido ajeno a esta "revolución de los datos", siendo actualmente un contribuyente significativo al ritmo de generación. Los datos de los que disponen las empresas no provienen exclusivamente de sus propios procesos internos (datos de inventario, ventas, financieros, etc.), de proveedores y competidores, siendo aspectos como la interacción con los clientes una fuente de generación cada vez mayor. Así, el *feedback* que se obtiene a través de los canales en las distintas redes sociales, las *reviews* de los productos de comercio electrónico, los *e-mails*, chats o llamadas referidas al servicio de atención al cliente son fuentes de un volumen de información cada vez mayor.

2.2 Big Data Analytics

Resulta innegable que el mundo se está digitalizando en, prácticamente, todos los ámbitos, desde la medicina hasta la administración, pasando por la empresa o la educación. Dicho nivel de penetración, unido a la disponibilidad masiva de datos debido a los factores previamente mencionados, tiene como consecuencia la aparición de todo un ecosistema en torno a la gestión de dichos datos (generación, procesado, almacenamiento, análisis, etc.), conocido como *Big Data*. De un modo general, se puede definir el *Big Data* como los conjuntos de datos que exceden ampliamente la capacidad del hardware y del software tradicionalmente utilizados en el ámbito de la tecnología de la información. Alternativamente, existe cierto consenso a la hora de abordar la definición del término *Big Data* desde la perspectiva de los datos, mencionando sus tres principales características, conocidas como las 3Vs [66] (volumen, velocidad y variedad), las cuales se describen a continuación:

- Volumen: Esta característica hace referencia al tamaño del volumen de datos que se genera y que debe ser recogido, almacenado y procesado. Es la característica más asociada al *Big Data*, puesto que el propio término así lo indica, pero no es suficiente para definir, en su totalidad, los conjuntos de datos.

- Velocidad: Los datos se generan a un ritmo muy alto, siendo en determinados ámbitos imprescindible su análisis con la mayor celeridad posible (tiempo real).
- Variedad: Los datos provienen de muchas y muy diversas fuentes, en formatos muy variados (fotos, audios, vídeos, texto, datos de GPS, etc.) y, en gran parte, no estructurados o semi-estructurados.

Las tres características anteriores forman la definición inicial asociada al *Big Data*. Con el tiempo, dicha definición se ha extendido añadiendo dos características adicionales, para formar el actual modelo conocido como de las 5Vs[1] [67]. Las dos características añadidas en último lugar han sido:

- Valor: Este término se refiere al valor, generalmente económico, que se puede obtener a través del análisis de los datos, ya sea haciendo a las empresas más productivas y competitivas o ayudando a la medicina en el diagnóstico y tratamiento de enfermedades.
- Veracidad: La calidad de los datos es un aspecto crucial, para evitar escenarios en los que los datos resultan contaminados por exceso de ruido. La proliferación masiva de cuentas controladas vía software en redes sociales son un claro ejemplo de la relevancia adquirida por esta característica.

A pesar del enfoque realizado en la singularidad de los datos para definir el *Big Data*, es importante destacar que los datos, por si solos, carecen de utilidad. Para extraer valor de éstos, deben ser analizados e interpretados con el fin de extraer información de ellos. Este proceso de análisis permite obtener información valiosa y útil, tales como patrones ocultos, preferencias de los clientes, tendencias de mercado, correlaciones desconocidas, etc., a partir de los conjuntos de datos, una tarea que se torna irrealizable para el ser humano dado el volumen y la complejidad de los mismos. Ejemplos significativos de lo que se puede conseguir a través del análisis de grandes conjuntos de datos incluyen el desarrollo de modelos de predicción de los brotes de virus de la gripe [68] (desarrollados por Google junto con el centro de control de enfermedades de los Estados Unidos), la gestión de contenidos multimedia a partir del análisis de hábitos y gustos en servicios de televisión a la carta [69] (Netflix, …) o la optimización en la logística en empresas de transporte [70] (tanto reparto como almacenamiento). Este proceso de obtención de información relevante a partir de conjuntos de datos, que responden al modelo de 5Vs, es comúnmente conocido como *Big Data Analytics*.

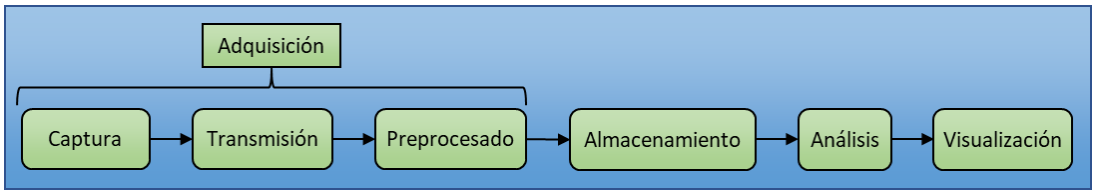

Figura 6 – Etapas del proceso de Big Data Analytics.

El análisis de datos a un nivel tan masivo se torna en un proceso complejo que impone requerimientos severos a las infraestructuras, tanto hardware como software, sobre las que se lleva a cabo. Las múltiples etapas en las que se divide el proceso de *Big Data Analytics* (Figura 6), incluyendo adquisición, almacenamiento, análisis y visualización de resultados [71] requieren de herramientas de trabajo específicas. Adicionalmente, la gestión y análisis de los enormes

[1] Algunos autores incluso van más allá de la definición de las 5Vs, añadiendo nuevos términos a la definición, tales como visibilidad, velocidad, validez, etc.

conjuntos de datos es una tarea con alta demanda computacional, por lo que la especialización hardware es también un requisito fundamental. Entre las etapas del proceso de análisis, cabe destacar la relevancia que han adquirido las herramientas de almacenamiento y organización de volúmenes de datos con las peculiaridades previamente descritas para este entorno. A lo largo del resto del capítulo se profundizará sobre los aspectos más destacables de este tipo de software.

2.3 Almacenamiento de Datos a gran Escala

El almacenamiento de volúmenes de datos a una escala sin precedentes requiere mecanismos altamente escalables, capaces de adaptarse al actual ritmo de generación de datos, donde la fiabilidad y la disponibilidad son dos requisitos fundamentales. La vía convencional para gestionar el creciente volumen de datos consistía en la inversión en hardware más potente, con mayores capacidades de almacenamiento y procesado (escalabilidad vertical). Desafortunadamente, el creciente coste asociado al escalado vertical ha hecho necesario poner el foco sobre modelos de computación alternativos. Los entornos de computación distribuidos basados en hardware de propósito general (*Cloud Computing*) proporcionan un modelo de gran eficiencia de cara a la escalabilidad horizontal (tanto en capacidad de almacenamiento como de cálculo), sustentando actualmente gran parte de los procesos de *Big Data Analytics* existentes. Estos entornos son capaces de gestionar aplicaciones cuyo volumen de datos puede exceder fácilmente el *petabyte* y de procesar tales datos en paralelo, reduciendo el coste computacional asociado [72]. Así, la relación existente entre el *Cloud Computing* y el *Big Data Analytics* ha supuesto y continuará suponiendo un impulso para ambos.

La necesidad de dotar a este tipo de entornos de procesos de almacenamiento altamente escalables, capaces de gestionar datos que responden al modelo de las 5Vs, ha potenciado el rápido desarrollo de *frameworks* orientados al almacenamiento y procesado distribuido de datos. Dos claros exponentes de este tipo de software son las bases de datos NoSQL [21], así como la librería Apache Hadoop [73]. Ambos son mecanismos de almacenamiento escalables horizontalmente sobre hardware de propósito general y distribuido, donde la elección de uno u otro depende de las necesidades concretas [74]. Así, por ejemplo, las bases de datos NoSQL son más apropiadas para el análisis en tiempo real de datos (s*treaming* de datos) o interactivo, cuyo volumen sea moderado, mientras que los modelos de programación basados en sistemas de ficheros distribuidos son más apropiados en entornos donde el tiempo no es un requerimiento estricto.

Hadoop es un claro ejemplo del software surgido para satisfacer las necesidades de almacenamiento y procesado del *Big Data*. Su origen se sitúa a principios de la década del 2000, dentro de un proyecto cuyo propósito era la creación de un motor de búsqueda en un escenario donde el volumen de sitios web ya suponía un enorme desafío [73]. Este *framework* tiene como objetivo el almacenamiento y procesado de grandes volúmenes de datos de manera distribuida sobre *clusters* con hardware de propósito general. Se compone, fundamentalmente, de dos elementos: un sistema de ficheros distribuido denominado *Hadoop Distributed File System* (HDFS) [75] y un modelo de programación denominado MapReduce [19]. El sistema de ficheros HDFS fue diseñado para su utilización en entornos distribuidos basados en hardware de propósito general. Además de las características comunes a cualquier sistema de ficheros distribuido, HDFS proporciona alta tolerancia a fallos o alto *throughput* de acceso a datos, ambas necesarias en entornos *Big Data*. Por su parte, el modelo de programación MapReduce se basa en el procesado de datos a través de dos funciones denominadas *map* y *reduce*. La primera de las dos funciones se encarga de dividir la tarea pertinente en sub-tareas más pequeñas y de asignar éstas a los

distintos nodos que componen el sistema, mientras que la segunda se encarga de recoger los resultados generados por las sub-tareas y de recomponerlos para obtener el resultado final. Con esta aproximación, el problema se puede resolver mediante el uso de gran cantidad de nodos con hardware de propósito general, obteniendo en conjunto una gran capacidad de procesamiento. Un ejemplo sencillo que ilustra el funcionamiento de MapReduce es el recuento del número de veces que aparece una letra en un conjunto de palabras (ver Figura 7). Hadoop está actualmente muy extendido, siendo utilizado por grandes compañías como la red social Facebook (mejoras en la experiencia de usuario a través de la recolección de estadísticas o lucha contra *spam*) [76], Ebay (mejora de experiencia de usuario y detección de fraude o robo de cuentas) [77], LinkedIn (recomendaciones) [78] o Yahoo (clasificación de imágenes) [79].

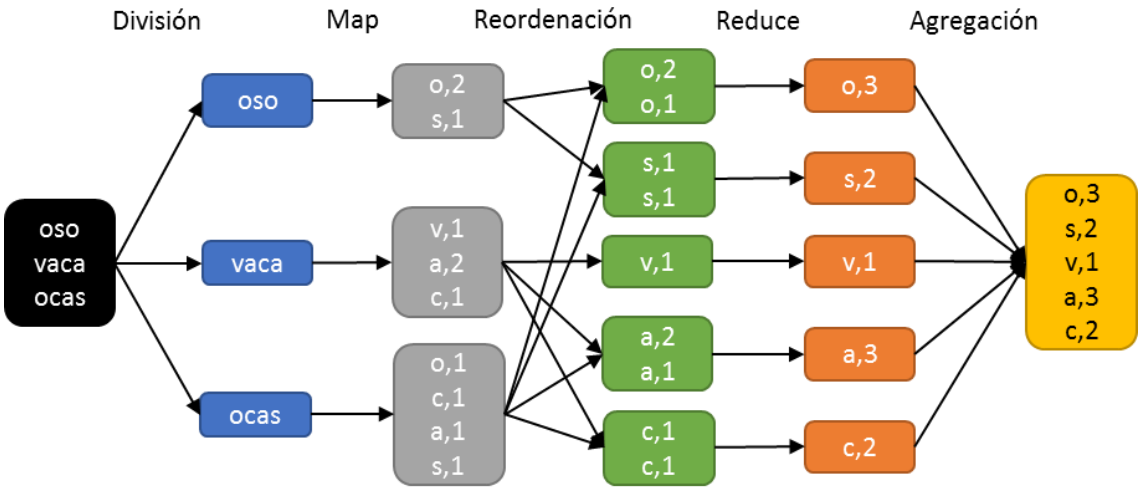

Figura 7 – Ejemplo del paradigma MapReduce. El conjunto de palabras se divide entre los nodos disponibles y a cada nodo se le asigna un subconjunto para que cuente las apariciones de cada letra (tuplas letra - nº apariciones). Los resultados de los nodos se reordenan en base a las claves de las tuplas (las letras) para poder así obtener el resultado final de cada letra. Finalmente, se obtiene el resultado total agregando los resultados de la reducción.

A pesar de su versatilidad, los entornos de cómputo distribuidos basados en Hadoop pueden ser insuficientes para dar respuesta a algunas de las peculiaridades del *Big Data Analytics*. Los requerimientos de rendimiento del análisis de datos en tiempo real (*streaming* de datos), el acceso interactivo a los datos o la importancia de las relaciones entre los datos hacen que el modelo de programación MapReduce y los sistemas de ficheros distribuidos no sean las herramientas más adecuadas en determinados entornos [80]. Recientemente, *frameworks* de procesado de datos capaces de proporcionar mayores cotas de rendimiento están ganando gran popularidad. Así, herramientas como Spark [20] consiguen desplazar a Hadoop en algunos casos. Haciendo uso de tecnologías de procesado en memoria (*in-memory*), Spark es capaz de ofrecer velocidades de procesado de datos significativamente más altas. De la misma forma, las bases de datos NoSQL ofrecen una solución orientada a mejorar las capacidades proporcionadas por mecanismos de almacenamiento basados en sistemas de ficheros. Estas bases de datos surgen como respuesta a las dificultades de las bases de datos tradicionales para trabajar con los conjuntos de datos del *Big Data* (en especial, debido a la variedad de los datos y a su volumen).

La popularidad de las bases de datos NoSQL no ha dejado de aumentar desde la pasada década (al igual que ha sucedido con Hadoop) y en la actualidad sus principales exponentes se sitúan entre las 10 primeras posiciones de la lista de las bases de datos más utilizadas globalmente [81]. Es más, a tenor de la evolución que han experimentado hasta el momento, resulta sensato esperar que su popularidad continúe aumentando (en paralelo a la expansión del *Big Data*) en los

próximos años, al igual que el volumen de negocio asociado [82]. Dicho éxito implica que las aplicaciones NoSQL se convertirán en software habitual en entornos *cloud*, lo que garantiza su ejecución sobre un número relevante de procesadores. Por este motivo, este tipo de aplicaciones debería formar parte de cualquier *suite* de *benchmarking* que quiera ser representativa de entornos de cómputo distribuidos. Actualmente, algunos *benchmarks* de reciente creación [27][83] incluyen entre sus cargas de trabajo alguna aplicación NoSQL. Desafortunadamente, en los casos mencionados se trabaja con una sola aplicación (Cassandra) y con un único patrón de acceso a los datos. Dada su relevancia, en esta tesis se ha realizado un esfuerzo significativo para trabajar con un conjunto mucho más amplio de este tipo de bases de datos y patrones de acceso, seleccionando un subconjunto que incluya los diferentes tipos existentes. Dada la relevancia de dichas aplicaciones, que serán una parte fundamental del software a utilizar para el desarrollo de esta tesis, se dedica la siguiente sección a proporcionar una introducción a este tipo de bases de datos, describiendo sus fundamentos, particularidades, clasificación, etc. Adicionalmente, se presenta el conjunto de bases de datos seleccionadas para el desarrollo de la tesis.

2.4 Bases de Datos NoSQL

En su estructura tradicional, las bases de datos implementadas sobre el modelo de datos relacional (RDBMS o *Relational Database Management System*) presentan dificultades para su adaptación a entornos de *Big Data Analytics*. Por un lado, la organización de datos en base a tablas funciona de manera correcta en presencia de datos altamente estructurados (cadenas de texto, números, fechas, etc.), cuya organización en filas/columnas resulta apropiada y facilita las labores de ordenación y procesado. Desafortunadamente, los volúmenes de datos en entornos *Big Data* carecen de estructuras claras y precisas en la mayoría de los casos (mezcla de ficheros de texto, *e-mails*, audios, videos, fotos, etc.) y en este contexto, tareas como la consulta o en indexado se vuelven altamente ineficientes [84]. Por otro lado, la escalabilidad de los sistemas RDBMS resulta compleja. Las estrictas condiciones impuestas sobre las transacciones con el fin de garantizar la consistencia de los datos almacenados dificultan en gran medida la escalabilidad horizontal (sistemas distribuidos) de las bases de datos RDBMS. Dicho conjunto de condiciones responde a las siglas ACID, que hacen referencia a cada una de sus propiedades (*Atomicy*, *Consistency*, *Isolation* y *Durability*):

- La atomicidad asegura que aquellas transacciones que involucran varias operaciones son atómicas, o se completan todas o la transacción entera falla, revirtiendo los datos a su estado original.
- La consistencia asegura que solamente se modifica la base de datos con datos válidos a través del cumplimiento de una serie de reglas de consistencia. De este modo, si una transacción no cumple alguna de las normas se revierte el estado de la base de datos, mientras que, si la transacción cumple las reglas, la base de datos pasa de un estado consistente a otro consistente.
- El aislamiento asegura que las transacciones concurrentes no interfieren entre sí, asegurando que el estado final de la base de datos es el mismo que si las transacciones tuvieran lugar secuencialmente.
- Por último, la durabilidad asegura que los datos se escriben en disco una vez la transacción se ha completado, evitando pérdidas de información en caso de un fallo en el sistema.

El cumplimiento de las reglas anteriores garantiza la fiabilidad de las bases de datos de forma rígida en caso de fallos hardware o imprevistos (como la caída del suministro eléctrico), pero, a su vez, dificulta en gran medida la escalabilidad horizontal del sistema. Ante este escenario, han surgido múltiples alternativas tratando de dar respuesta a los nuevos condicionantes, tanto de escalabilidad como de modelo de datos, agrupados bajo la etiqueta de bases de datos NoSQL (*no SQL* en su acepción original y *Not Only SQL* actualmente). A pesar de que el término tiene su origen en la década de los 90 [85], su reaparición en relación a entornos *Big Data* es más reciente, con exponentes como BigTable (Google, 2004) [86], Amazon Dynamo (Amazon, 2007) [87] o Cassandra (Facebook, 2008) [22], entre los más destacados.

Con el fin de eliminar la rigidez del modelo ACID para las transacciones, las bases de datos NoSQL relajan esos requisitos en favor de la escalabilidad implementando un conjunto de requisitos alternativos denominado BASE (*Basic Availability, Soft-state, Eventual consistency*). Así, mientras que ACID es pesimista forzando a mantener la consistencia tras cada transacción, BASE es optimista y acepta inconsistencias temporales. Con la implementación de BASE, se logra facilitar la escalabilidad horizontal, pudiendo desplegar las bases de datos sobre cientos o miles de nodos, con una capacidad de almacenamiento agregada de cientos de *petabytes*. Sus principales propiedades se definen a continuación:

- *Basic availability*: Los datos está disponibles la mayor parte del tiempo, obteniendo un alto grado de disponibilidad gracias la replicación, aunque los datos puedan ser temporalmente inconsistentes.
- *Soft-state*: El estado de la base de datos puede cambiar en el tiempo incluso sin modificaciones por parte de los usuarios debido a la consistencia eventual.
- *Eventual consistency*: En ACID, se asegura la consistencia tras cada transacción, mientras que en BASE el sistema será consistente en algún momento. Los datos se copiarán a todos los nodos correspondientes eventualmente, de tal modo que no se asegura que los datos estén actualizados en un momento dado.

El modelo de datos de estas nuevas bases de datos no está basado en el modelo relacional, además, su diseño facilita la escalabilidad y las dota de la capacidad de trabajar con datos cuya naturaleza es no estructurada. A diferencia de las bases de datos relacionales, diseñadas para trabajar con datos altamente estructurados en forma de tablas con filas y columnas, las nuevas bases de datos contemplan un abanico más amplio de datos acorde a las características de los conjuntos de datos del *Big Data,* que incluyen datos no estructurados, semi-estructurados y estructurados. El amplio conjunto de bases de datos NoSQL se clasifica, generalmente, en base al modelo de datos utilizado, definiendo así las cuatro categorías siguientes:

- **Clave-valor**: Las bases de datos clave-valor son el tipo de base de datos NoSQL más sencillo. No presentan esquema alguno, al contrario que en el modelo relacional, dado que los datos se organizan en una colección de parejas clave-valor (ver Figura 8). La clave de cada pareja es única y el valor puede ser cualquier dato (un entero, un *string*, un documento JSON (*JavaScript Object Notation*), una imagen, un vídeo, etc.), en definitiva, el modelo de datos de este tipo de bases de datos es un *array* asociativo (como un diccionario o un mapa). En general, este tipo de bases de datos no tiene un lenguaje de consulta, sino que el acceso a los datos se lleva a cabo de una manera muy simple, en base a unos pocos comandos (insertar, obtener, borrar, etc.), donde siempre es necesario especificar una determinada clave para operar con el valor. Los datos son opacos (dado que se almacenan como objetos binarios), y, por tanto, no es posible filtrar los resultados de las peticiones en base a su

contenido. Además, la modificación de los datos implica la sustitución completa del valor asociado a una determinada clave. Debido a su sencillez, son mucho más rápidas y fácilmente escalables [88] (en base a particionar y replicar el contenido) que las bases de datos relacionales. Redis [89] y Voldemort [90] son dos ejemplos de este tipo de bases de datos.

Clave	Valor
Clave_1	110100010101110
Clave_2	10111010011010011
Clave_3	11110010001101
Clave_4	100110011

Figura 8 – Ejemplo del modelo de datos de las bases de datos clave-valor.

- **Orientadas a documentos**: Las bases de datos orientadas a documentos llevan a cabo el almacenamiento y la consulta de los datos en base a documentos, tal y como indica su nombre. Son similares a las bases de datos clave-valor, con la salvedad de que el valor de cada pareja clave-valor se restringe a datos semi-estructurados en forma de documentos (ver Figura 9). Éstos responden generalmente a un formato standard como XML (*eXtensible Markup Language*), YAML (*YAML Ain't Markup Language*), JSON o BSON (*Binary JSON*). Cada documento define su estructura y almacena los datos a través de una serie de parejas nombre-valor, y aunque el contenido suele seguir una estructura similar, no tiene por qué ser idéntica. El esquema es, por tanto, muy flexible, es posible añadir nuevos datos a un documento en particular, sin tener que modificar el esquema de toda la base de datos. A diferencia de las bases de datos clave-valor, donde los datos son opacos, las bases de datos orientadas a objetos permiten hacer búsquedas aplicando filtros en base al contenido de los datos, así como modificaciones de los datos sin tener que sustituirlos por completo. Dos ejemplos de estas bases de datos son MongoDB [23] (BSON) y CouchDB [91] (JSON).

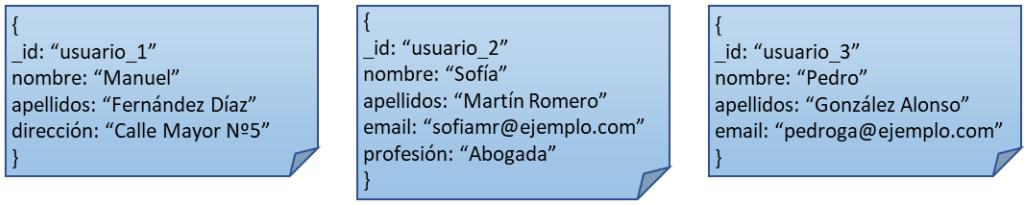

Figura 9 – Ejemplo del modelo de datos de las bases de datos orientadas a documentos.

- **Orientadas a grafos**: Este tipo de bases de datos está optimizado para datos que están muy conectados entre sí y, para lograrlo, basa su modelo de datos en la teoría de grafos, almacenando la información y sus relaciones en base a nodos y aristas (ver Figura 10). Los nodos representan entidades, almacenan la información, son el equivalente a una fila de una tabla del modelo relacional o a un documento de las bases de datos orientadas a documentos. Por su parte, las aristas conectan unos nodos con otros, representando las relaciones existentes entre los datos. Aunque el modelo relacional es capaz de capturar los datos y sus relaciones (mediante la creación de tablas adicionales para las relaciones y costosas operaciones *join* para la consulta de los datos), resulta complejo e ineficiente lograrlo cuando los datos están muy conectados. Por el contrario, las bases de datos basadas en grafos permiten capturar los datos y sus complejas interacciones, así como la exploración de la red, obteniendo un buen rendimiento en consultas que implican explorar

tales relaciones. En general, este tipo de base de datos es adecuado cuando los datos están muy conectados y las consultas implican la exploración de las relaciones entre los datos. Varios ejemplos de este tipo de bases de datos son OrientDB [92], neo4j [93] o ArangoDB [94].

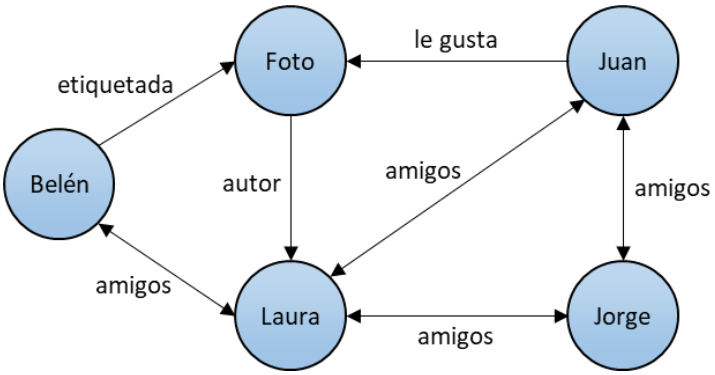

Figura 10 – Ejemplo del modelo de datos de las bases de datos orientadas a grafos.

- **Orientadas a columna**: Las bases de datos orientadas a columna (también conocidas como *wide-column*) son muy eficientes a la hora de llevar a cabo tareas como el particionado o la compresión de los datos, y son, además, altamente escalables. Utilizan un modelo de datos similar al de las bases de datos relacionales, está basado en tablas con filas y columnas (ver Figura 11). No obstante, al no haber un esquema como en el modelo relacional, los datos no tienen por qué ser idénticos en cada fila y es posible modificar su estructura en cualquier momento. La unidad básica de los datos que utiliza se denomina "columna" y consiste en una pareja nombre-valor. Además, se pueden definir "supercolumnas", otra unidad de datos similar (de tipo clave-valor) cuyo campo valor es un mapa que contiene un número ilimitado de columnas (no de supercolumnas). Así, las filas (identificadas mediante una clave única) contienen una serie de columnas (o supercolumnas), tantas como sean necesarias para el almacenamiento de los datos pertinentes, y se pueden agrupar en lo que se denominan *column-families y supercolumn-families*. Estas estructuras deben declararse antes de la creación de los datos y albergan un número ilimitado de filas que se almacenan de forma conjunta en el disco. El contenido de las filas que forman parte de estas estructuras tampoco tiene por qué ser idéntico, de tal manera que el número de columnas o supercolumnas o el identificador de las mismas pueden variar de una fila a otra. Las dos bases de datos más populares orientadas a columna son Cassandra [22] y Hbase [95].

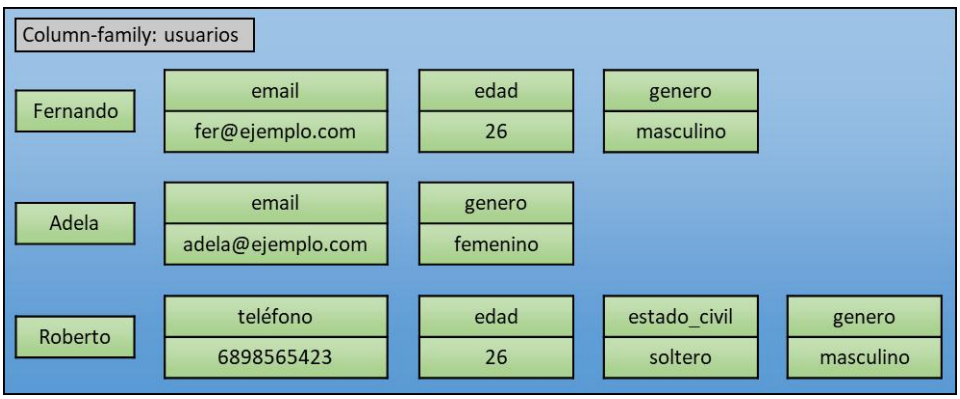

Figura 11 – Ejemplo del modelo de datos de las bases de datos orientadas a columna.

La creciente relevancia de los entornos *Big Data Analytics* presenta un claro exponente en la cantidad y diversidad de bases de datos NoSQL surgidas en los últimos años. Dada su extensión, el objetivo en la presente tesis ha sido utilizar un representante de cada uno de los modelos de datos expuestos en la clasificación previa. Así, todos los procesos de *profiling*, caracterización y evaluación realizados a lo largo de los próximos capítulos utilizarán Redis [89] como representante del modelo clave-valor, MongoDB [23] representando al modelo basado en documentos, OrientDB [92] en representación del modelo orientado a grafos y Cassandra [22] como la base de datos representante del modelo orientado a columnas.

2.4.1 Redis

Redis es una base de datos *in-memory* cuya primera versión pública data de mayo de 2009. Está escrita en ANSI C y distribuida bajo licencia BSD (*Berkeley Software Distribution*). Se trata de un software de gran versatilidad, pues además de como base de datos, puede ser utilizado como *cache* para otras aplicaciones o como *message broker (*programa intermediario que traduce los mensajes de un sistema desde un lenguaje a otro). Es la base de datos de tipo clave-valor más utilizada [81], algunos ejemplos de compañías que utilizan esta base de datos son Twitter, GitHub y StackOverflow.

Debido al modelo de datos que implementa Redis, los datos son opacos y, además, no hay esquema alguno. El tipo de dato que se utiliza tanto para la clave como para el campo valor de cada pareja es el *string*. Éste es una cadena de bytes que puede contener cualquier tipo de dato y cuyo tamaño máximo se restringe a 512MB. Así, las claves pueden ir desde una simple cadena de caracteres hasta una imagen en formato JPEG (*Joint Photographic Experts Group*), y, en el caso de los valores, se puede utilizar simplemente un *string*, o estructuras de datos más complejas tales como listas, *sets*, etc., cuyo elemento básico es el *string*.

Redis utiliza la memoria principal para el almacenamiento de los datos (en vez de un dispositivo de almacenamiento no volátil), lográndose así que sea extremadamente rápida. Sin embargo, este diseño implica que no es fiable, puesto que los datos se pierden en su totalidad ante un imprevisto como un fallo en el hardware o la interrupción del suministro eléctrico. Para evitar o minimizar la pérdida de datos soporta dos mecanismos de persistencia: *snapshotting* y *append-only file.* El primer mecanismo se basa en salvar el estado de la base de datos al completo a través de la creación de *snapshots* (ficheros con el contenido de la base de datos) en el disco, de tal forma que es posible restaurar la base de datos a partir de ellos, si bien todos los cambios acontecidos tras la creación de un *snapshot* no están asegurados. Además, es posible automatizar su creación o realizarla manualmente a través de comandos disponibles. El segundo mecanismo se basa en el registro de las operaciones que modifican el contenido de la base de datos en un fichero de *log*, de tal modo que es posible reconstruir la base de datos desde cero aplicando secuencialmente cada uno de los comandos registrados en caso de que sea necesario, y cuya granularidad es configurable entre salvar cada operación o salvar el conjunto de operaciones cada segundo. Dado que se almacena cada operación que modifica el estado, el tamaño de este fichero no deja de aumentar progresivamente (no se almacena el valor actual sino todas las operaciones que han llevado a ese valor actual), afortunadamente, Redis es capaz de reescribir este fichero minimizando el número de operaciones. La vía más adecuada para la protección de los datos pasa por la utilización de ambos mecanismos conjuntamente.

Redis es fácilmente escalable y elástica, permite añadir o eliminar nodos de manera sencilla, adecuando las necesidades de almacenamiento y procesado sin provocar la interrupción del servicio de la base de datos. El particionado de los datos entre los nodos disponibles es a nivel de

clave y, en general, no permite transacciones ni operaciones que involucren múltiples claves. Se lleva a cabo en base a la utilización de una función *hash* para determinar en qué nodo se mapean los datos, siendo necesario migrar parte de éstos cuando se añade o elimina un nodo, y además gestionar múltiples ficheros para la asegurar la persistencia de los datos. Implementa un modelo de replicación basado en maestros y esclavos, muy fácil de utilizar y de configurar, con el que consigue gran disponibilidad y mejorar el rendimiento. En este modelo, cada nodo maestro puede tener varias réplicas exactas (pudiendo configurarse para ser de solo lectura), y cada réplica puede ser a su vez el maestro de otros nodos esclavos, definiendo así una estructura en forma de árbol. Los datos del nodo maestro se propagan a los esclavos de manera asíncrona y no bloqueante, de tal manera que el maestro puede continuar atendiendo peticiones en caso de que se esté produciendo una sincronización completa o parcial, aunque también es posible forzar una replicación síncrona desde el cliente, utilizando un determinado comando.

2.4.2 MongoDB

MongoDB es una base de datos multiplataforma orientada a documentos y diseñada para lograr un alto rendimiento, una gran disponibilidad y una fácil escalabilidad. Es una base de datos de código abierto, escrita en C++ y cuyo lanzamiento tuvo lugar en el año 2009. Expedia, Forbes, Bosch o Cisco son ejemplos de compañías que utilizan esta base de datos.

MongoDB guarda los datos en documentos cuyo formato es BSON, una representación binaria del formato JSON y por tanto no legible, que además extiende los tipos de datos recogidos en el formato JSON. La estructura de este tipo de documentos es una serie de parejas nombre-valor (ver Figura 12), que no tiene por qué ser idéntica en todos los documentos (esquema flexible), donde el nombre es un *string* y el valor uno de los distintos tipos de datos soportados por el formato. Los documentos se pueden agrupar en colecciones, el equivalente a una tabla del modelo relacional, independientemente de su estructura interna. El tamaño máximo de los documentos está fijado en 16MB con el objetivo de limitar la cantidad de memoria principal utilizada a la hora de realizar una consulta, dado que el documento se carga al completo. Cada registro tiene un identificador único para su identificación (clave primaria) y que siempre está en la primera posición. En el caso en el que un documento no contenga este identificador a la hora de su inserción en la base de datos se crea automáticamente, dándole como valor una marca de tiempo, lo que permite ordenar posteriormente los documentos. El formato soporta distintos tipos de datos como *strings*, números o booleanos, así como datos más complejos incluyendo *arrays*, documentos BSON o *arrays* de documentos.

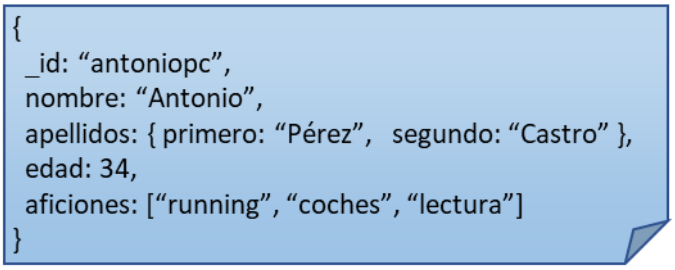

Figura 12 – Ejemplo del formato de los documentos.

La alta disponibilidad de los datos, así como la redundancia, se logran a través de la replicación. Ésta se basa es la ejecución de múltiples instancias del servicio *mongod* en una serie de nodos que contienen una copia idéntica de los datos y que se organizan en lo que se denomina *replica set*. En estas réplicas, el rol de los nodos no es idéntico, sino que cada una está compuesta por un nodo

primario, varios nodos secundarios y, opcionalmente, un árbitro. En caso de que el nodo primario pase a no estar disponible (los nodos se monitorizan entre sí mediante el intercambio de mensajes), se desencadena un proceso de elección para promocionar a un nodo secundario al rol de primario. El rol de los árbitros está, precisamente, relacionado con este proceso, ya que su función no es mantener una réplica de los datos y dar servicio, sino desempatar la elección de un nuevo primario, asegurando que el número de votos de la réplica sea impar. Las escrituras se restringen al nodo primario, propagándose a los nodos secundarios de manera asíncrona y las operaciones de lectura también se dirigen por defecto al nodo primario. No obstante, los clientes tienen la posibilidad de dirigir las operaciones de lectura a los nodos secundarios, aunque, en ese caso, los datos que obtienen pueden no estar actualizados debido a la replicación asíncrona.

MongoDB escala horizontalmente particionando (*data sharding*) los datos entre los nodos disponibles, logrando así aumentar la capacidad y el rendimiento. La arquitectura distribuida de esta base de datos se organiza en torno a lo que denomina *sharded clusters* (ver Figura 13), los cuales están compuestos por *shards*, *mongos* y *config servers*.

- Los *shards* son los nodos que contienen los datos. Cada *shard* almacena una parte de los datos asignados al *sharded cluster* y se puede desplegar como *replica set* para asegurar los datos.
- La función de los nodos *mongos* es ser la interfaz entre los clientes y el *sharded cluster*, de tal manera que las aplicaciones no se comunican directamente con los *shards*. Actúan como un *router*, dirigiendo las consultas y las operaciones de escritura al *shard* correspondiente, utilizando para ello la información (metadatos) de los servidores de configuración.
- Los *config servers* almacenan los metadatos del *sharded cluster*. Éstos reflejan el estado y la organización de los datos y de los componentes del *cluster*, incluyendo qué datos están en cada *shard*. En este caso, es necesario su despliegue como *replica set*.

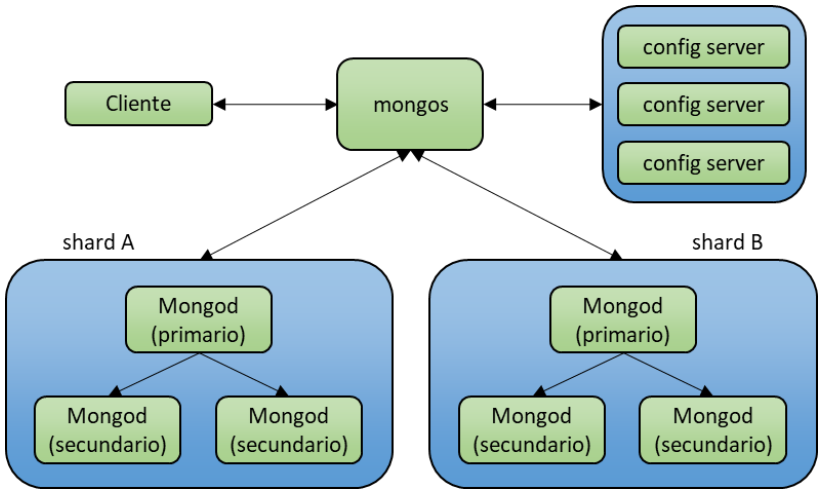

Figura 13 – Modelo de particionado de MongoDB.

2.4.3 OrientDB

OrientDB es una base de datos NoSQL multi-modelo, es decir, soporta distintos modelos de datos (grafos, documentos, clave-valor y objetos), pero las relaciones entre los datos se implementan como en una base de datos orientada a grafos nativa, con relaciones directas entre los datos. Es capaz de asegurar ACID en las transacciones, al igual que el modelo relacional, y utiliza SQL como

lenguaje de consulta, aunque extiende su funcionalidad para trabajar con grafos. Su lanzamiento tuvo lugar en el año 2010, bajo licencia de software libre y, actualmente, es una de las bases de datos orientadas a grafos más utilizadas. Entre sus clientes destacan Dell, Cisco, Sky o las Naciones Unidas.

En OrientDB, la unidad más pequeña que se puede consultar o almacenar se denomina *registro*, identificándose cada uno de manera única, y se implementan cuatro tipos: documentos, vértices, aristas y registros binarios (como en las bases de datos clave-valor). Cada registro (a excepción de los binarios) suporta distintos tipos de datos, tales como números (varios formatos), booleanos, cadenas de caracteres, fechas, registros embebidos, etc. La definición del contenido de cada registro se basa en el concepto de clase (heredado del paradigma de programación orientado a objetos) para definir los datos (nombre y tipo) y determinadas reglas, soportando tres esquemas distintos: *schema-full*, *schema-mixed* y *schema-less*. OrientDB utiliza el concepto de *cluster* aplicado a las clases, de tal modo que es posible particionar los registros pertenecientes a una determinada clase y almacenarlos, por ejemplo, en distintos discos. Con ello se consigue optimizar las consultas, puesto que es posible consultar, únicamente, los datos de un único *cluster*, además de aumentar el rendimiento a través de consultas en paralelo sobre datos almacenados en distintas localizaciones.

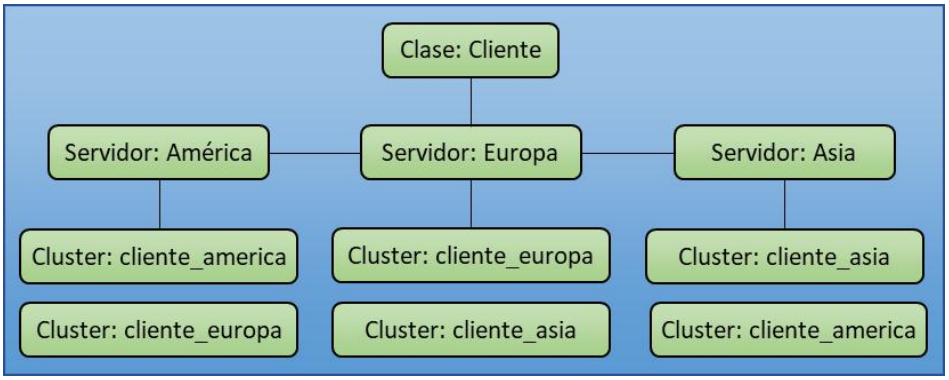

Figura 14 – Modelo de replicación y particionado de OrientDB.

OrientDB escala horizontalmente en base al particionado y a la replicación (ver Figura 14), aplicándose ambos al concepto de *clusters* y clases. La replicación que implementaba originalmente era maestro-esclavo, pero fue sustituida debido al cuello de botella existente con las escrituras por el modelo *multi-master.* En este modelo, todos los nodos aceptan tanto operaciones de lectura como de escritura, aunque también es posible la configuración de nodos como réplicas de solo lectura, y se elimina por tanto el cuello de botella con las operaciones de escritura del primer modelo. El particionado se soporta a nivel de clase, utilizando múltiples *clusters* por cada clase, de tal modo que los diferentes *clusters* se mapean en distintos nodos. Así, cada *cluster* (datos) puede residir en uno o más nodos, aplicándose replicación en el último caso. La Figura 14 muestra una arquitectura donde se aplican de manera conjunta las técnicas de particionado y replicación. En este caso particular, los datos de la clase *Cliente* se particionan en 3 *clusters* (denominados *Europa*, *América* y *Asia*) y se distribuyen entre los 3 nodos disponibles. Además, se replican los datos con un factor 2, de tal modo que cada *cluster* está presente en dos servidores.

2.4.4 Cassandra

Cassandra es actualmente la base de datos orientada a columnas más utilizada [81], con grandes compañías como Netflix, Ebay o Reddit destacando entre un gran número de usuarios. Está escrita

en el lenguaje de programación Java y fue desarrollada inicialmente por Facebook, compañía que decidió lanzarla como proyecto de software libre en el año 2008, siendo actualmente responsabilidad de la fundación de software Apache. En general, esta base de datos resulta adecuada para proyectos que requieran una enorme escalabilidad, gran disponibilidad y tolerancia a fallos. Su diseño aúna el modelo de datos de BigTable [86] y la arquitectura distribuida de Amazon Dynamo [87], y destaca por ofrecer un rendimiento sobresaliente en las operaciones de escritura. Define su propio lenguaje de consulta, denominado CQL (*Cassandra Query Language*), y soporta distintos tipos de datos (varios formatos de números, cadenas de texto o secuencias de bytes), permitiendo además la definición de nuevos tipos de datos a través de CQL. Asimismo, integra Hadoop (la implementación libre del paradigma MapReduce), donde el rol de Cassandra es proporcionar los datos para que Hadoop los procese, además del almacenamiento de los resultados.

Cassandra es capaz de escalar horizontalmente en base a una arquitectura distribuida y elástica que permite particionar y replicar los datos (ver Figura 15), consiguiendo así que tanto el rendimiento de las operaciones (lectura y escritura) como la capacidad escalen linealmente a medida que se añaden o eliminan nodos. En el momento en el que se produce una variación en el número de nodos, se hace necesaria la migración de parte de los datos, pero sin que este proceso provoque la interrupción del servicio. Esta arquitectura es descentralizada y se asemeja a una topología *token ring*, donde todos los nodos tienen el mismo rol (no hay maestros ni esclavos) y aceptan peticiones de lectura y escritura. En esta arquitectura, los clientes pueden enviar operaciones de lectura o escritura a cualquier nodo del sistema ya que éstos actúan como coordinadores, es decir, actúan como un *proxy* entre el cliente y el nodo o nodos donde residen los datos.

Implementa dos estrategias de replicación (una para despliegues con un único *datacenter* y otra para entornos con varios), permitiendo así almacenar y gestionar réplicas de los datos en los distintos nodos que componen el sistema para garantizar la fiabilidad y la tolerancia a fallos. En ambas estrategias, es posible configurar el número de réplicas, lo que se denomina factor de replicación. De este modo, la utilización del valor 1 para el factor de replicación mantiene una única copia de cada fila, mientras que un factor de replicación con un valor X (mayor que 1) mantiene X réplicas de cada fila. En este último caso, todas las réplicas tienen la misma importancia, es decir, no hay réplicas maestras. La replicación permite que la base de datos funcione con normalidad en el caso de que uno o varios nodos no estén disponibles, así como la reconstrucción de los datos de un nodo desde cero, siempre y cuando haya suficientes réplicas vivas en el anillo. Además, los distintos niveles de consistencia implementados, tanto para las lecturas como paras las escrituras, pueden configurarse a nivel global, a nivel de *cluster* o incluso especificarse en cada petición por parte de los clientes.

La partición de los datos (incluyendo las réplicas) entre los distintos nodos disponibles se implementa en base a la utilización de una función *hash* sobre las claves de las filas (independientemente del tipo de dato utilizado para la clave) que minimiza la reorganización de los datos en el momento en el que se añade o se elimina un nodo. El rango de la función *hash* se divide entre el número de nodos disponibles, asignando a cada nodo un identificador y una porción del rango, lo que permite distribuir los datos y determinar, en todo momento, en qué nodo reside un dato en particular (ver Figura 15 (arriba)). Además, se implementa otro paradigma de particionado denominado “nodos virtuales”, el cual permite la asignación de múltiples rangos por nodo. Es análogo al anterior, el rango de la función de *hash* se divide y soporta replicación, pero las porciones son más pequeñas y su número es mayor que el número de nodos disponibles,

de tal modo que se asignan múltiples regiones por nodo (ver Figura 15 (abajo)). Con la utilización de nodos virtuales se automatiza el cálculo y asignación de los rangos, así como el recálculo necesario tras la adición o eliminación de un nodo. La utilización de este paradigma facilita, además, el uso sistemas heterogéneos, adecuando el particionado al rendimiento de cada máquina mediante la asignación un número distinto de rangos a cada máquina en función de las características particulares.

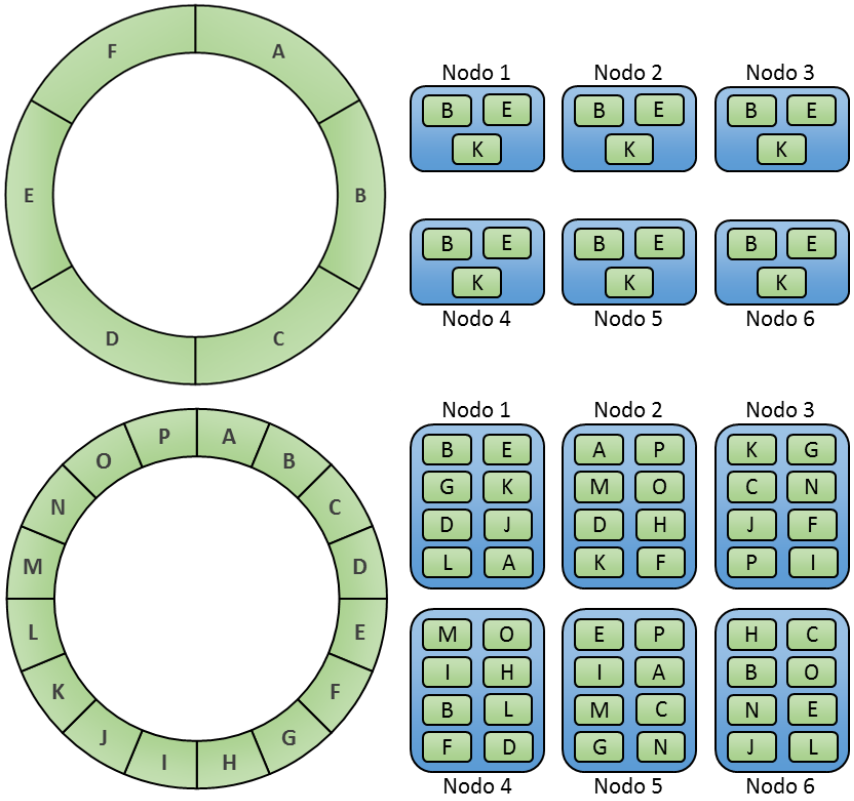

Figura 15 – Ejemplo de la arquitectura con: (arriba) 6 nodos, 6 rangos y un factor de replicación 3 (abajo) 6 nodos, 16 rangos y un factor de replicación 3.

2.5 Especialización SW frente a HW de Propósito General

La expansión y el éxito del *Big Data* se debe al esfuerzo llevado a cabo en el desarrollo de software específico para trabajar con las características de los conjuntos de datos. Se trata de un software que tiene actualmente una gran cuota de mercado y que continúa en expansión, siendo cada vez más numerosas las empresas que optan por soluciones basadas en *Big Data* para incrementar su competitividad. Es previsible que el volumen de negocio asociado a este tipo de herramientas, como soporte a conjuntos de datos de tipo *Big Data*, continúe aumentando notablemente en los próximos años, justificando su inclusión en *benchmarks* actuales.

En general, este tipo de software se ejecuta en entornos de *Cloud Computing*, donde la capacidad agregada de múltiples *datacenters* (compuestos por un elevado número de nodos) permite satisfacer los requerimientos de almacenamiento. Ante la creciente demanda, la mejora de eficiencia de cómputo para este tipo de entornos adquirirá cada vez más importancia para los proveedores de servicios (Amazon, Google, etc.) [96]. Paradójicamente, en toda esta evolución, los procesadores a cargo de ejecutar todo este software han sufrido poca o nula especialización. La arquitectura de los procesadores orientados a servidor es similar a la de los procesadores utilizados en entornos de escritorio, tratándose de hardware de propósito general con una

arquitectura que busca maximizar el ILP (*Instruction-Level Parallelism*) (la cantidad de instrucciones independientes de una aplicación que pueden ser ejecutadas simultáneamente). Para ello, esta arquitectura implementa mecanismos como la ejecución fuera de orden o la ejecución especulativa, que requieren de estructuras complejas que ocupan mucha área y consumen mucha energía (tales como el ROB (*Re-Order Buffer*), el *scheduler* o la cola de *loads*/*stores*). A pesar de su consideración de propósito general, aquellas aplicaciones incapaces de explotar el ILP de manera adecuada infrautilizarán los recursos del procesador. Ante esta situación, parece conveniente evaluar si las aplicaciones pertenecientes al software del *Big Data* hacen un uso efectivo de estructuras hardware que fueron ideadas para software con poco o nada que ver con entornos *Big Data*. De esta manera será posible detectar las posibles ineficiencias de las arquitecturas actuales, sentando las bases para poder desarrollar soluciones arquitecturales más acordes a sus características, con el objetivo de incrementar el rendimiento y reducir el consumo energético.

Adicionalmente, desde el punto de vista de la metodología de investigación, las nuevas bases de datos representan un desafío para las herramientas de simulación dadas algunas de sus particularidades, como su *stack* software o su arquitectura cliente-servidor. Estas características fuerzan a las herramientas de simulación a ser capaces de simular sistemas multi-nodo, posibilitando la correcta evaluación del software del servidor al evitar la interacción cliente-servidor sobre los recursos hardware si ambos se ejecutasen en un único nodo. De la misma forma, el incremento de complejidad del *stack* software (en comparación con aplicaciones tradicionalmente utilizadas para labores de *benchmarking* [53]) hacen más relevante la necesidad de evaluar el efecto del sistema operativo sobre el rendimiento, dado que la fracción de código privilegiado deja de ser despreciable. Todos estos requerimientos no hacen sino incrementar el coste computacional asociado al proceso de simulación.

En el resto de la presente tesis se analizan y proponen un conjunto de requerimientos mínimos para considerar la metodología de trabajo como adecuada para evaluar el tipo de aplicaciones emergentes que se han descrito en el presente capítulo. Se comienza detallando el trabajo necesario sobre la herramienta de simulación, analizando la metodología más adecuada para estas aplicaciones y describiendo el *framework* utilizado por el grupo de trabajo, adaptado para trabajar en entornos multi-nodo y con aplicaciones NoSQL. Dicho *framework* ha posibilitado los siguientes pasos del trabajo desarrollado, consistentes en el estudio y la caracterización con gran nivel de detalle de la eficiencia de dichas aplicaciones en el uso de los componentes de la jerarquía de memoria del procesador, cuyos resultados se muestran el capítulo 4. Finalmente, y gracias de nuevo al *framework* y a las metodologías propuestas, el capítulo 5 cierra la tesis conjugando aplicaciones y tecnologías emergentes, evaluando propuestas arquitectónicas con tecnología de memoria no volátil sobre un amplio conjunto de *benchmarks*.

3 Metodología de Trabajo

3.1 Introducción

El capítulo anterior ha puesto de manifiesto la rápida evolución del software que es posible encontrar actualmente en cualquier entorno en el que se haga uso de un computador. En un esfuerzo por seguir incrementando las cotas de rendimiento, los trabajos de investigación en el área de Arquitectura de Computadores deben adaptar su metodología a dicha evolución. Es necesario destacar que esta área de investigación presenta peculiaridades que condicionan la metodología aplicable a cualquier proceso de evaluación de una nueva propuesta. Por un lado, la extrema complejidad del proceso de fabricación de los procesadores actuales hace imposible la fabricación de prototipos como parte del proceso de evaluación. Esta limitación fuerza la utilización de metodologías basadas en la simulación software de componentes hardware, así como enfoques centrados en el análisis exhaustivo de las aplicaciones a través de herramientas de *profiling* hardware. Por otro lado, es harto complicado prever de manera precisa qué tipo de software se ejecutará en un procesador durante su ciclo de vida útil, haciendo necesario recurrir a herramientas de estandarización de software conocidas como *benchmarks*.

El presente capítulo se centra en la metodología de trabajo en el área de Arquitectura de Computadores y se puede dividir en dos grandes bloques. El primer bloque, correspondiente con la sección 3.1, presenta las herramientas de *profiling*, simulación y *benchmarking* habituales del área, describiendo a continuación su utilización conjunta como *framework* de trabajo. El segundo bloque (secciones 3.2 y 3.3) se centra en analizar las peculiaridades de las aplicaciones emergentes, así como sus efectos sobre las metodologías de evaluación más habituales.

3.1.1 *Profiling* con Contadores Hardware

En los últimos años, una parte creciente de los trabajos en el área de Arquitectura de Computadores hace uso de herramientas de *profiling* implementadas sobre los contadores de eventos micro-arquitecturales presentes en la mayoría de los procesadores modernos (ver Figura 3). Dicho hardware permite la monitorización de eventos durante la ejecución de aplicaciones, y, en consecuencia, posibilita el análisis del rendimiento de las aplicaciones sobre el hardware subyacente. Este análisis no solo es útil para poder diseñar arquitecturas según las demandas de las aplicaciones, sino que también permite que los desarrolladores de software puedan optimizar el código de sus aplicaciones para extraer el máximo rendimiento del sistema.

El hardware utilizado para la monitorización de eventos, denominado *Performance Monitoring Unit* (PMU), implementa la lógica que controla un reducido grupo de registros (*Event Counters*) utilizados para contabilizar la ocurrencia de eventos, siendo dicha cuenta de eventos configurable a través de una serie de registros adicionales denominados *Performance Event Select Registers* (PESR). En su mayor parte, dichos eventos se centran en aspectos relativos al *core* (su *pipeline* de ejecución y la jerarquía de memoria), pero en las últimas familias de procesadores se extienden también a eventos que tienen lugar fuera del *core* (*uncore* PMU). Aspectos como el número de contadores implementados, los eventos disponibles para su medida o el formato de los registros PESR varían en función del fabricante e incluso en función del modelo de procesador para un mismo fabricante. A modo de ejemplo, la Figura 16 muestra la implementación de un registro PESR para las arquitecturas Intel. Dicho registro incluye campos para la definición del evento a

monitorizar (*Event Select* + *Unit Mask*), así como campos que controlan el modo de contabilizar los eventos (en modo usuario (USR) o modo privilegiado (OS), cuenta inversa (INV), etc.)

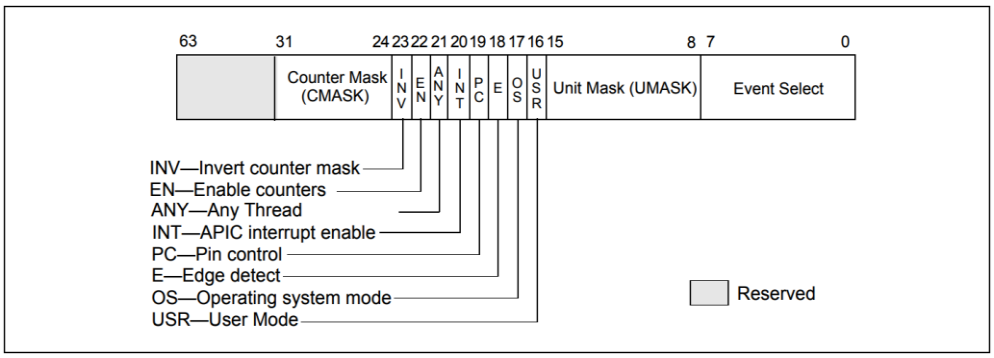

Figura 16 – Formato de un registro PESR de Intel [97].

La programación de los contadores es un proceso delicado que puede realizarse directamente mediante instrucciones propias de cada ISA (*Instruction Set Architecture*) que permiten configurar los registros PESR. Desafortunadamente, el número de eventos monitorizables en un procesador moderno es, en general, muy grande (pudiendo sobrepasar la centena) y la documentación para identificar dichos eventos es, en ocasiones, sumamente compleja, provocando que la "curva de aprendizaje" sea larga y pudiendo llevar a situaciones en las que resulta difícil decidir qué eventos monitorizar para obtener la información pertinente. Afortunadamente, existen herramientas que facilitan dicha tarea, abstrayendo el uso de los contadores del hardware subyacente, tales como *perf* [98] o *vtune* [99]. Este tipo de herramientas define un conjunto de eventos simbólicos, iguales para cualquier arquitectura soportada (*perf* realiza la traducción etiqueta – contador hardware) correspondientes con aquellos eventos hardware más comúnmente utilizados (un evento simbólico puede hacer referencia a aspectos micro-arquitecturales que implican múltiples eventos hardware).

Perf es una de las herramientas cuyo uso está más extendido, siendo actualmente parte del paquete de software *Linux-tools-common* de las distribuciones de dicho sistema operativo. Su utilización se extiende más allá de la cuenta de eventos hardware, ofreciendo múltiples utilidades de *profiling*. A través de *perf*, es posible utilizar los contadores hardware de dos modos distintos: cuenta de eventos y muestreo de aplicaciones. En modo "cuenta", los contadores se programan y monitorizan la aplicación durante el periodo de medida, tras lo que se obtiene la cuenta agregada de eventos. En cambio, en modo "muestreo", los contadores se programan para volcar información sobre la ejecución (estado de los registros, contador de programa, etc.) cada vez que un evento determinado alcanza un valor previamente definido para proceder, posteriormente, a su análisis. La cuenta de eventos a través de esta herramienta es altamente configurable, pudiendo realizar múltiples medidas de manera simultánea (teniendo en cuenta la limitación impuesta por el número de registros implementados) o definir el ámbito de la medida (por procesador, por proceso, a nivel global) entre otros aspectos.

Tomando como base este tipo de herramientas, han surgido recientemente mecanismos de análisis de rendimiento para la identificación de cuellos de botella en procesadores con ejecución OoO (*Out-of-Order*), posibilitando que los arquitectos de computadores puedan comprender los recursos que demandan las aplicaciones emergentes. El principal exponente es *Top-Down* [100], una metodología que permite identificar los principales cuellos de botella de las arquitecturas Intel de un modo correcto y rápido. El análisis que propone se lleva a cabo mediante la monitorización de determinados eventos (definidos por la metodología) siguiendo una estructura

jerarquizada (ver Figura 17), siendo responsabilidad del usuario la determinación de la región a analizar. Esta monitorización se centra en el flujo de instrucciones que tiene lugar en dos puntos concretos del *pipeline* de ejecución: en el *issue* (la frontera entre el *Front-End* y el *Back-End*) y en el *retire*. Idealmente, el flujo de instrucciones en cada ciclo en ambos puntos debería ser igual a la anchura del *pipeline*, de tal manera que mediante la utilización de los contadores hardware es posible determinar el origen de la disminución de ambos flujos. El análisis se realiza de manera iterativa y, gracias a su estructura jerárquica, se centra únicamente en aquellas ramas que, sucesivamente, se van marcando como relevantes hasta llegar a los niveles más profundos.

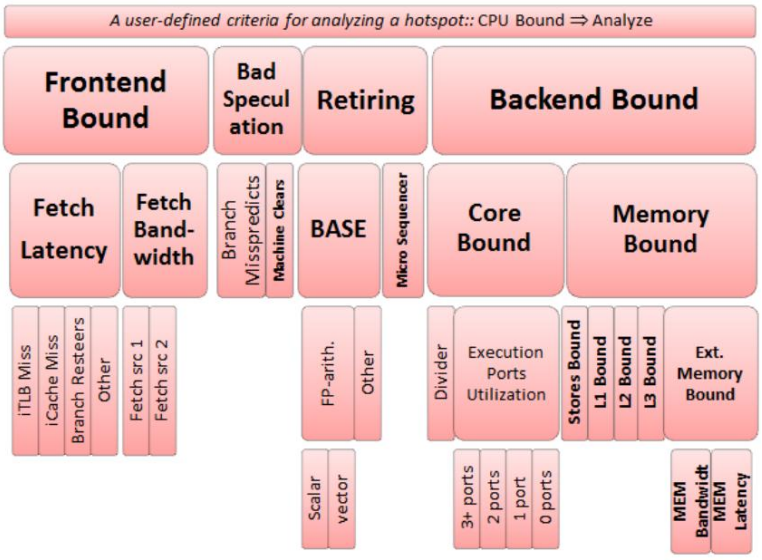

Figura 17 – Organización jerárquica de las métricas de la metodología Top-Down [100].

Las metodologías basadas en la utilización de contadores hardware proporcionan un mecanismo de evaluación cuyo coste computacional es muy eficiente, puesto que la velocidad de ejecución de las aplicaciones es nativa, permitiendo el análisis de las aplicaciones sobre fracciones de tiempo elevadas. Sin embargo, uno de los principales inconvenientes de este tipo de metodologías es su rigidez. La reconfiguración o modificación del hardware subyacente no es posible y, por tanto, no permite la implementación y el estudio de propuestas arquitectónicas. Por esta razón, este tipo de metodologías se limita a tareas de evaluación de rendimiento y detección de cuellos de botella, recayendo la labor de elaborar y evaluar mecanismos correctores sobre herramientas de simulación, habituales en esta área de trabajo.

3.1.2 Simulación

La simulación es la metodología de investigación más habitual en el área de Arquitectura de Computadores (ver Figura 3). Las herramientas de simulación convierten el hardware en un modelo software, dotándolo de flexibilidad y haciéndolo fácilmente parametrizable. Así, la exploración del espacio de diseño o la evaluación de una propuesta arquitectónica se tornan posibles a través de la modificación del código y/o de los parámetros del simulador y de la ejecución de determinados *benchmarks*. La complejidad de la herramienta de modelado hardware varía enormemente entre el gran número de simuladores disponibles [55], desde la simulación únicamente de un elemento arquitectural aislado mediante trazas de ejecución (como un predictor de saltos) [55], hasta un sistema completo (procesador + jerarquía de *cache*, memoria principal, almacenamiento y I/O) capaz de ejecutar el *stack* software sin modificar (sistema operativo + aplicaciones) [55]. La principal limitación para la utilización de herramientas de simulación precisas es el coste computacional asociado, puesto que la simulación es una tarea

lenta en comparación con la ejecución sobre el hardware real. La velocidad de simulación varía en función de la amplitud del hardware modelado y del nivel de detalle, pero incluso en aquellos escenarios con un nivel de detalle limitado no es posible llevar a cabo evaluaciones con la misma metodología utilizada cuando se trabaja con hardware real [101] (no es posible simular las aplicaciones de principio a fin en un tiempo razonable). Asimismo, resulta extremadamente complejo modelar hasta el último detalle de las arquitecturas modernas, puesto que no solo aumentaría notablemente la complejidad de los simuladores, sino que, además, extendería aún más el tiempo de ejecución de los trabajos de simulación. Por estos motivos, las metodologías basadas en simulación siempre presentarán cierto margen de error con respecto al hardware real, dependiendo su valor del compromiso entre precisión y coste adoptado.

La precisión y la complejidad de estas herramientas dependen en gran medida de dos aspectos importantes de la simulación: del *timing* y de la funcionalidad de las instrucciones. El término *timing* hace referencia a la simulación de las latencias de los distintos elementos de la micro-arquitectura del procesador, tales como la jerarquía de *cache* o el *pipeline* de ejecución. Así, los simuladores que lo modelan (ya sea parcial o completamente) simulan por ejemplo el retardo que provocan las latencias de las peticiones a memoria o el paso de las instrucciones a través de las distintas etapas del *pipeline* de ejecución, mientras que, en aquellas herramientas que no lo modelan, las instrucciones se ejecutan sin ningún tipo de retardo. Como es lógico, el modelado del *timing* implica que el modelo del hardware de la herramienta es más preciso, pero al mismo tiempo más complejo, y viceversa. Por otra parte, la simulación de la funcionalidad de las instrucciones hace referencia al modelado de la semántica de las instrucciones que forman parte del ISA. Las herramientas que simulan la funcionalidad modelan la ejecución de las instrucciones tal y como sucede en el hardware real, es decir, tomando valores de entrada y generando valores de salida (los cuales son leídos y almacenados en los registros del procesador) y moviendo además estos datos entre la jerarquía de memoria y el procesador. Dentro de amplio abanico de simuladores, existen aquellos que modelan la funcionalidad por sí mismos, los que delegan estas tareas en otras herramientas y los que no la modelan y recurren a trazas de ejecución como se verá a continuación. Al igual que sucede con el *timing*, la funcionalidad contribuye notablemente a incrementar la precisión y la complejidad.

Dentro de las herramientas de simulación disponibles, aquellas dirigidas por trazas (*trace-driven simulation*) [55] ofrecen la versión más sencilla. En este tipo de herramientas se simula únicamente el *timing* del hardware específico que se modela, no la funcionalidad de las instrucciones. Estas herramientas no son capaces de simular la ejecución de una aplicación, sino que es necesario pre-procesar dicha aplicación para obtener trazas de su ejecución, que son utilizadas como valores de entrada. El contenido de dichas trazas se corresponde con una secuencia ordenada de eventos (referencias a memoria, predicciones de instrucciones de salto, etc.) correspondiente con las partes del hardware modeladas por la herramienta y se obtiene bien a través del uso de software específico (software de instrumentación o emuladores) [102][103] o bien a través del hardware (mediante técnicas de monitorización) [102][103]. En general, estas herramientas son más sencillas de implementar y más rápidas en el proceso de simulación, debido a que no modelan la funcionalidad de las instrucciones. Sin embargo, esta metodología presenta limitaciones importantes respecto a la precisión de los resultados obtenidos. La ejecución conducida por trazas es incapaz de capturar el efecto de algunos de los mecanismos hardware habituales en los procesadores actuales, tales como la ejecución especulativa (predicción de saltos o ejecución fuera de orden) o las relaciones dinámicas entre múltiples *threads* (captura de *locks* para la exclusión mutua) que se producen en las aplicaciones paralelas [55]. Asimismo, estas

herramientas imponen un orden de ejecución determinista (el capturado en la traza) y no contemplan las alteraciones producidas por los cambios del hardware en el orden de ejecución.

Actualmente, la simulación conducida por trazas está cayendo en desuso frente a metodologías más precisas como la simulación conducida por ejecución [55] (*execution-driven simulation*). Este tipo de herramientas son más complejas, dado que simulan tanto la ejecución de las aplicaciones (la funcionalidad de las instrucciones), como el *timing* del hardware. Algunas de estas herramientas llegan a un nivel de precisión tan alto que son capaces de ejecutar un sistema operativo sin modificar y ejecutar sobre éste las aplicaciones pertinentes [55]. En este tipo de entornos, la precisión de las evaluaciones realizadas alcanza un nivel relevante, reduciendo significativamente las diferencias con entornos de hardware real [104][105]. No obstante, el alto nivel de precisión puede poner en tela de juicio la representatividad de los resultados, debido a que únicamente es posible simular una pequeña parte de las aplicaciones. A pesar de sus elevados requerimientos computacionales, el acceso cada vez mayor de los investigadores a infraestructuras de cálculo de gran capacidad, así como la utilización de múltiples técnicas que acotan los tiempos de simulación [106][107] han convertido este tipo de herramientas en las más extendidas. Finalmente, cabe mencionar que a medio camino entre la ejecución por trazas y la simulación de sistema completo, existen también aproximaciones intermedias que no incluyen el sistema operativo, emulando en este caso el funcionamiento de las "llamadas al sistema" [108].

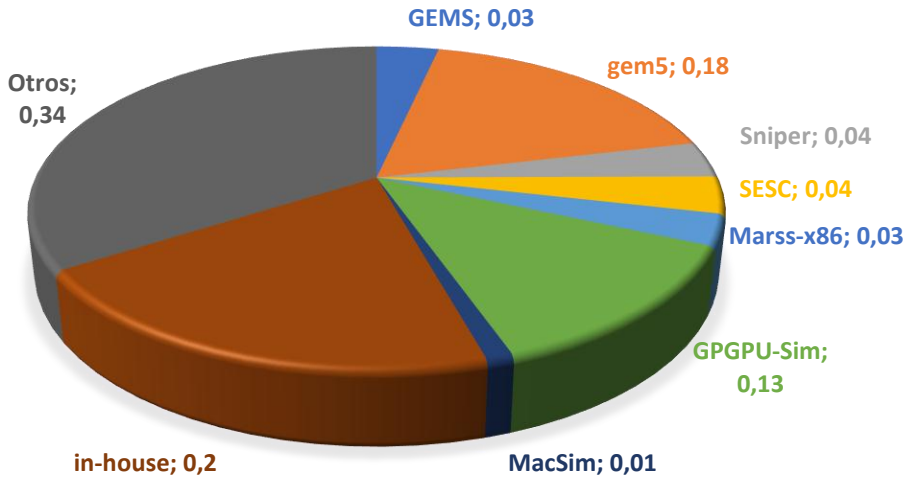

Figura 18 – Exploración de los simuladores utilizados en el área de Arquitectura de Computadores.

La Figura 18 muestra la elevada heterogeneidad en las herramientas de simulación empleadas de acuerdo con la exploración realizada en el Capítulo 1. Se incluye a continuación una breve descripción de aquellos simuladores con un nivel de utilización significativo, correspondiendo con aquellos que alcanzan, al menos, un valor del 3% en la citada exploración.

- **GEMS** (*General Execution-driven Multiprocessor Simulator*) [109] es un simulador compuesto por dos módulos que se acoplan al simulador Simics [110], logrando, de manera conjunta, la simulación detallada de sistemas multiprocesador. El módulo de procesador (*Opal*) implementa parcialmente el ISA SPARC v9 y modela el *timing* de una CPU (*Central Processing Unit*) con ejecución fuera de orden, similar al MIPS R10000 [111], mientras que el módulo *Ruby* hace lo propio con el sistema de memoria (*caches*, redes de interconexión, coherencia, etc.). Por su parte, Simics es un simulador funcional de sistema completo sobre el que se puede ejecutar un sistema operativo sin modificaciones y su rol dentro de esta herramienta es la simulación de la funcionalidad de las instrucciones.

- **Gem5** [108] es el resultado de la unión de dos simuladores: m5 y GEMS. Hereda el modelo de CPU de m5 y el modelo de memoria de GEMS. Este simulador implementa varios modelos de CPU, con diferente nivel de detalle, desde una arquitectura superescalar sin modelado de *timing* hasta una arquitectura con ejecución fuera de orden y especulativa con modelado del *timing*. De la misma forma, también conviven en gem5 modelos de memoria con diferente nivel de detalle, uno más sencillo y rápido heredado del simulador m5 y Ruby, heredado del simulador GEMS. Soporta dos modos de simulación: *full system* (sistema operativo + *benchmarks*) y *system-call emulation*. La mayoría de los ISAs comerciales están soportados (Alpha, ARM, SPARC, MIPS, Power, RISC-V y x86), lográndolo a través del desacople de su semántica y de los modelos de CPU.

- **Sniper** [112] es una herramienta de simulación que está construida sobre Graphite [113], un simulador *multicore* (ISA x86) distribuido, que se basa, a su vez, en Pin [114]. Sniper implementa un modelo de procesador basado en *interval simulation* [115], una técnica de simulación con un nivel de abstracción mayor que la simulación ciclo a ciclo utilizada comúnmente y cuya complejidad es mínima (alrededor de 1.000 líneas de código) en comparación con los modelos de CPU detallados. Con la utilización de esta técnica se consigue que la herramienta sea rápida y, por tanto, apropiada para la exploración de grandes sistemas *multicore* (con hasta cientos de *cores*) en un tiempo limitado, donde la precisión no es el factor más importante.

- **SESC** (*SuperESCalar Simulator*) [116] es un simulador capaz de modelar una arquitectura actual con elementos como ejecución fuera de orden, predicción de saltos o jerarquía de *cache on-chip*, los cuales permiten alcanzar un nivel de detalle muy alto. Se limita al ISA de MIPS y es capaz de llevar a cabo tanto simulaciones con arquitecturas de procesador con un único *core*, como arquitecturas con múltiples *cores*. Se trata de un simulador dirigido por eventos que separa la funcionalidad del *timing*, lo que facilita tanto el desarrollo del código como el *debugging*. Las instrucciones son ejecutadas por un módulo de emulación que está basado en el emulador MINT (MIPS) [117] y que genera cierta información por cada instrucción ejecutada (dirección de la instrucción, registros fuente y destino o unidades funcional utilizada) que se le pasa al módulo de simulación para que modele el *timing* de la micro-arquitectura.

- **Marssx86** (*Micro-ARchitectural and System Simulator for x86-based Systems*) [118] es un simulador de sistema completo restringido al ISA x86_64 y basado en el simulador PTLSIM [119]. Permite realizar simulaciones *single-core* o *multicore* y provee dos modos de simulación: una arquitectura con ejecución en orden y otra con ejecución fuera de orden. Está integrado con QEMU [120], de tal modo que es posible alcanzar un punto de interés de la aplicación a gran velocidad utilizando QEMU y comenzar en ese punto la simulación con uno de los dos modos de simulación soportados. Permite simular un sistema operativo junto con las aplicaciones pertinentes sin ninguna modificación, y proporciona soporte para *debugging*.

- **GPGPU-Sim** [121] es un simulador de GPUs que permite ejecutar *benchmarks* paralelizados con CUDA u OpenGL. Soporta el ISA Nvidia PTX y modela con gran detalle la arquitectura y el *timing* de una GPU de la arquitectura Nvidia Kepler.

- **MacSim** [122] es un simulador de arquitecturas heterogéneas dirigido por trazas. Soporta los ISAs x86 y Nvidia PTX, y se apoya en sendas herramientas para la generación las trazas: Pin [114] para la CPU y GPUOcelot [123] para la GPU. Modela con detalle *pipelines* con

ejecución en orden y fuera de orden, además del sistema de memoria (*caches*, NoC (*Network-on-Chip*) y controladores de memoria), y permite simulaciones con configuraciones asimétricas de los *cores*.

Profundizando sobre los resultados mostrados en la Figura 18, es destacable la gran cantidad de trabajos publicados que hacen uso de una herramienta *in-house* (simuladores desarrollados por los autores de los trabajos y cuyo código fuente no está accesible públicamente) así como el tamaño de la categoría "otros" (simuladores utilizados por un número reducido de autores), sumando entre ambas categorías el 54%. Resulta cuando menos curioso que en más de la mitad de los trabajos presentados a este tipo de conferencias, consideradas de primer nivel, se trabaje con metodologías y herramientas que dificultan o directamente imposibilitan la reproducción de los resultados obtenidos. La exploración realizada también revela que un creciente número de investigadores opta por herramientas de libre disposición y con un modelo de desarrollo abierto como es el simulador *gem5*, siendo el más utilizado de aquellos disponibles públicamente.

Gem5 ha sido la herramienta de simulación utilizada para el desarrollo de las tesis por múltiples motivos. Se trata de una herramienta con un alto grado de precisión que es capaz de llevar a cabo simulaciones de un sistema completo y destaca como la más completa (implementa distintos modelos de CPU y memoria, ISAs, modos de simulación, etc.). Cuenta con una comunidad de usuarios que ofrece soporte y que continúa con su desarrollo de manera muy activa. Finalmente, el grupo de investigación posee amplia experiencia tanto con esta herramienta de simulación como con GEMS (su predecesora), contribuyendo en ambos casos al desarrollo de dichos simuladores a través de la inclusión de un simulador de redes *on-chip* desarrollado en el seno del grupo [124] para su uso con GEMS o gem5. Con la utilización de estas herramientas el grupo ha sido capaz de llevar a cabo contribuciones en múltiples mecanismos del subsistema de memoria, tal y como atestiguan los trabajos publicados en protocolos de coherencia [125], la red de interconexión *on-chip* [126], los controladores de memoria [127], y los algoritmos de reemplazo [128][129].

3.1.3 *Benchmarking*

Las crecientes necesidades de las aplicaciones a ejecutar sobre cualquier procesador hacen del software el elemento clave en cualquier proceso de evaluación arquitectural, independientemente de la metodología utilizada. Desafortunadamente, evaluar las implicaciones de cualquier propuesta hardware sobre todo el software que ejecutará durante su ciclo de vida útil es una tarea impracticable. Es, por tanto, necesario, limitar la cantidad de software a emplear en la evaluación para poder realizarla en una cantidad de tiempo razonable, asegurando al mismo tiempo que un número demasiado pequeño de aplicaciones no degraden la validez de los resultados. Afortunadamente, existen herramientas de evaluación diseñadas con ese motivo, utilizando conjuntos de aplicaciones que son representativas de un conjunto mucho más amplio y denominados *benchmarks*.

Dado que el computador se ha convertido en una herramienta de trabajo universal, existe una gran diversidad de *suites* de aplicaciones que pretenden ser representativas de entornos muy diversos. Dicha heterogeneidad hace complejo llevar a cabo cualquier intento de clasificación. En este apartado se proporciona una breve descripción de aquellas *suites* más comúnmente utilizadas en el área, escogiendo el entorno de ejecución como elemento definitorio para la clasificación. Así, se definen cinco grandes categorías:

- **Entorno de escritorio:** este tipo de *benchmark* incluye aplicaciones muy heterogéneas, pertenecientes a campos como el análisis financiero, *data mining*, procesado de imagen,

etc. Dichos *benchmarks* intentan recoger software cuyo uso es habitual en equipos utilizados en entornos domésticos y laborales. Las *suites* de *benchmarks* más utilizadas para la evaluación de este hardware son SPEC [53] y PARSEC [54]. SPEC CPU 2006 (reemplazada recientemente por la nueva versión de 2017) incluye un conjunto de 29 aplicaciones secuenciales (12 de enteros y 17 de punto flotante) escritas en distintos lenguajes de programación (C, C++ y fortran). PARSEC es una *suite* complementaria a la anterior que permite la evaluación de entornos de escritorio desde la perspectiva de las aplicaciones paralelas. Incluye un total de 13 aplicaciones paralelas escritas en C/C++.

- **HPC (*High Performance Computing*)**: Este tipo de aplicaciones pertenecen históricamente al ámbito de la física y la ingeniería, tales como la física de partículas, la dinámica de fluidos, la química o la astronomía. Las aplicaciones HPC se caracterizan por tener unos requerimientos computacionales y de memoria que sobrepasan ampliamente las capacidades de los entornos de escritorio, haciendo que el entorno de ejecución apropiado sean los supercomputadores. Algunos de las *suites* de *benchmarks* más destacables son Linpack [130], Graph500 [131], NPB [132] o HPCC [133]. Los *benchmarks* Linpack y Graph500 se utilizan de manera conjunta para la elaboración de las listas top500 y graph500. Linpack evalúa la capacidad de punto flotante con un algoritmo para la resolución de sistemas de ecuaciones lineales, mientras que graph500 estresa el sistema de comunicaciones. Por su parte, la *suite* NPB (*NAS Parallel Benchmark*) reúne aplicaciones desarrolladas por la NASA, tratando de imitar las características computacionales y los movimientos de datos de aplicaciones del ámbito de la dinámica de fluidos. Finalmente, HPCC (*HPC Challenge*) es un conjunto compuesto por 7 aplicaciones que evalúan tanto capacidad de cómputo en punto flotante (en simple y doble precisión) como la latencia y el ancho de banda a memoria de los supercomputadores resolviendo problemas de álgebra de matrices.

- **GPUs (*Graph Processing Units*)**: La arquitectura de las GPUs, compuestas por cientos o miles de *cores* sencillos, las hace adecuadas para la ejecución de tareas que requieran gran número de operaciones sencillas y altamente paralelizables. Diseñadas en su origen para el procesamiento gráfico, su uso se ha extendido a la ejecución de aplicaciones del ámbito de la ingeniería o científico, que tradicionalmente se han ejecutado sobre procesadores de propósito general. Actualmente, es un área de investigación muy activa y entre las *suites* de *benchmarks* disponibles destacan Rodinia [134], Parboil [135] y Lonestar [136]. Rodinia es un conjunto de más de 20 aplicaciones para sistemas heterogéneos, incluyendo dominios como la física, el *data mining*, el procesado de imágenes o la bioinformática. Parboil es una *suite* similar a Rodinia que emplea modelos de programación basados en CUDA y OpenCL. Finalmente, Lonestar es otra *suite* de *benchmarks* para GPGPU (*General-Purpose computing on Graphics Processing Units*) que, a diferencia de las dos anteriores, contiene aplicaciones paralelas irregulares, aquellas cuyos datos no se almacenan en forma de matrices sino en estructuras de árbol o grafo.

- ***Cloud Computing* (*Big Data*)**: En los últimos años se ha desarrollado mucho software novedoso a raíz del surgimiento del *Big Data* y del desarrollo del *Cloud Computing*. Han surgido, en consecuencia, *benchmarks* que permiten la evaluación del hardware en base a este tipo de aplicaciones. Algunos ejemplos de *suites* representativas de este software son CloudSuite [27], BigDataBench [28], YCSB [137] y HiBench [138]. Cloudsuite es un conjunto de *benchmarks* que incluye una amplia variedad de aplicaciones representativas de este entorno, desde aplicaciones de análisis de datos (MapReduce [19], con técnicas *in-memory*

basadas en Apache Spark [20]) hasta servicios propios de entornos *cloud* como buscadores web, *streaming* y almacenamiento (base de datos NoSQL Cassandra). BigDataBench es una *suite* similar a la anterior, que incluye actualmente más de 40 aplicaciones entre *benchmarks* y *microbenchmarks*, así como conjuntos de datos reales (provenientes de Wikipedia, Amazon o Facebook). Con un enfoque más reducido, YCSB (*Yahoo Cloud Benchmark Suite*), se centra exclusivamente en aplicaciones de almacenamiento, proporcionando una interfaz para evaluar una amplia variedad de bases de datos NoSQL. Finalmente, HiBench es otra *suite* que se centra en la categoría de procesado de datos (*Big Data*), concretamente en Hadoop, la implementación *open-source* del paradigma MapReduce.

3.2 *Framework* de Trabajo: Punto de Partida

El trabajo enmarcado en esta tesis se ha llevado a cabo conjugando la herramienta de simulación (gem5) y *benchmarks* representativos de diferentes entornos de ejecución. Se dedica esta sección del capítulo a la herramienta de simulación para poner en valor el esfuerzo que requiere el desarrollo de nuevas propuestas con herramientas altamente complejas como la utilizada. El desarrollo de propuestas a través de gem5 implica una "curva de aprendizaje" lenta, y no son abundantes los trabajos sustentados por evaluaciones de sistema completo con este simulador. Esta sección describe en detalle cada uno de los *benchmarks* tradicionales empleados en las diferentes propuestas, explicando el proceso de adaptación llevado a cabo para su ejecución en un entorno simulado. Adicionalmente, se describe de manera precisa la metodología de simulación empleada para obtener los resultados que se presentan a lo largo del resto del capítulo.

3.2.1 *Benchmarking* Tradicional: SPEC, PARSEC y NPB

Los *benchmarks* SPEC, PARSEC y NPB han sostenido gran parte de la investigación en el área de Arquitectura de Computadores desde hace varios años. En la actualidad, siguen siendo muy utilizados a pesar de ser un software con más de diez años de antigüedad (ver Figura 2). La investigación realizada en el grupo en el que se enmarca esta tesis ha estado utilizando cada uno de estos *benchmarks* de manera habitual y también van a formar parte de esta tesis.

Tabla 1 – SPEC INT 2006.

Aplicación	Lenguaje	Descripción
401.bzip2 (BZ)	ANSI C	Compresión mediante bzip.
403.gcc (GC)	C	Compilador de C.
429.mcf (MC)	ANSI C	Optimización para horarios de transporte público.
445.gobmk (GO)	C	Ejecuta el juego denominado Go.
456.hmmer (HM)	C	Búsqueda de secuencias genéticas mediante modelos ocultos de Markov.
458.sjeng (SJ)	ANSI C	Programa de ajedrez.
462.libquantum (LI)	C99	Simulación de computación cuántica.
464.h264ref (H2)	C	Compresión de video.
471.omnetpp (OM)	C++	Simulación mediante OMNet++ de una extensa red Ethernet.
473.astar (AS)	C++	Algoritmo A* para obtención de rutas en mapas 2D.
483.xalancbmk (XA)	C++	Transforma documentos XML mediante Xalan-C++.

Las aplicaciones incluidas en la *suite* de *benchmarks* SPEC CPU2006 se categorizan en aquellas que trabajan utilizando tipos de datos enteros y aquellas que utilizan tipos de datos de punto flotante. Debido a la naturaleza secuencial de estas aplicaciones, se utiliza una estrategia de replicación, creando así mezclas homogéneas donde cada *core* del CMP (*Chip Multi-Processor*) ejecuta una

copia de la misma aplicación. Las aplicaciones se compilan utilizando la versión 4.7.2 de gcc, g++ y gfortran (nivel de optimización O3) y se utiliza el tamaño de problema denominado *reference* (el más grande que se proporciona). La Tabla 1 y la Tabla 2 recogen respectivamente las aplicaciones seleccionadas de la categoría de enteros y de punto flotante.

Tabla 2 – SPEC FP 2006.

Aplicación	Lenguaje	Descripción
410.bwaves (BW)	Fortran	Dinámica de fluidos computacional.
416.gamess (GA)	Fortran	Computación química.
433.milc (MI)	C	Cromodinámica cuántica.
434.zeusmp (ZE)	Fortran	Dinámica de fluidos computacional.
435.gromacs (GR)	C/Fortran	Simulación de dinámica molecular.
436.cactusADM (CA)	C/Fortran	Resuelve las ecuaciones de Einstein utilizando el método numérico de salto escalonado.
437.leslie3d (LE)	Fortran	Dinámica de fluidos computacional.
444.namd (NA)	C++	Simulación de Sistema bio-moleculares.
447.dealII (DW)	C++	Resuelve ecuaciones diferenciales parciales utilizando el método de elementos finitos.
450.soplex (SO)	C++	Programación lineal (método Simplex).
453.povray (PO)	C++	Renderizado de imagen.
454.calculix (CA)	C/Fortran	Mecánica estructural.
459.GemsFDTD (GE)	Fortran	Electromagnetismo computacional (Ecuaciones de Maxwell).
470.lbm (LB)	ANSI C	Dinámica de fluidos (método de Lattice Boltzmann).
481.wrf (WR)	C/Fortran	Predicción del tiempo.
482.sphinx3 (SP)	C	Reconocimiento de voz.

PARSEC (versión 3.0) incluye una serie de aplicaciones *multithread* muy diversa, con aplicaciones seleccionadas de áreas muy diferentes (la Tabla 3 recoge una breve descripción de cada una). Las aplicaciones están programadas haciendo uso de los lenguajes de programación C y C++, y se implementan distintos modelos de paralelización: POSIX *threads* (Pthreads), OpenMP e *Intel Threading Building Blocks* (TBB). La *suite* incluye una implementación en base a Pthreads para la gran mayoría de las aplicaciones (salvo *freqmine*), pero solo unas pocas implementan OpenMP o TBB. Se definen distintos tamaños de problema para realizar simulaciones micro-arquitecturales y, además, un tamaño para ejecutar las aplicaciones sobre hardware real, que es mucho más grande en comparación con el resto. Se selecciona el tamaño más grande destinado a simulación, denominado *simlarge*. Las aplicaciones se compilan utilizando las versiones 4.7.2 de gcc y g++ con un nivel de optimización O3, y se utiliza el modelo de paralelización por defecto de cada aplicación.

Tabla 3 – Aplicaciones de la suite PARSEC 3.0 seleccionadas.

Aplicación	Paralelización	Descripción
Blackscholes (BL)	Pthreads C	Calcula el valor de una opción de mercado utilizando la ecuación parcial diferencial Black-Scholes.
Bodytrack (BO)	Pthreads	Visión computerizada.
Facesim (FS)	Pthreads	Simula la física del rostro humano generando animaciones realistas.
Ferret (FE)	Pthreads	Motor de búsqueda para imágenes, audio, vídeo, etc.
Fluidanimate (FA)	Pthreads	Dinámica de fluidos para animación.
Freqmine (FM)	OpenMP	Implementación del método FP-growth para *data mining*.
Raytrace (RT)	Pthreads	Renderizado 3D de escenas animadas.
Swaptions (SQ)	TBB	Precio de una cartera de este tipo de derivados financieros.
Vips (VI)	Pthreads	Procesado de imagen.
X264 (X2)	Pthreads	Codificación de video.
Canneal (CA)	Pthreads	Simulador para optimizar el coste de un diseño de chips.
Dedup (DE)	Pthreads	Compresión de un flujo de datos.
Streamcluster (SC)	Pthreads	*Clustering online* de un flujo de entrada.

NPB se compone de una serie de aplicaciones paralelizadas con OpenMP (versión 3.2) (también implementa MPI). Define distintos tamaños de problema (denominándolos clases) clasificándolos como *standard* y *large*, de los cuales se selecciona la "B", el tamaño medio de los categorizados como *standard*. De nuevo, se compila el código utilizando la versión 4.7.2 de gcc y gfortran, con un nivel de optimización "O3". La Tabla 4 contiene una breve descripción de las aplicaciones que se van a utilizar, así como el lenguaje empleado para su implementación.

Tabla 4 – Aplicaciones de la suite NAS Parallel Benchmark (versión 3.3) seleccionadas.

Aplicación	Lenguaje	Descripción
BT	Fortran	Resolución de un sistema de ecuaciones en derivadas parciales mediante la factorización en tres operandos.
CG	Fortran	Gradiente conjugado para la resolución de un sistema de ecuaciones lineal.
FT	Fortran	Solución de una ecuación diferencial mediante transformadas rápidas de Fourier.
IS	C	Ordenación de una lista de número enteros mediante *bucket sort*.
LU	Fortran	Resolución de un sistema de ecuaciones en derivadas parciales mediante un algoritmo de factorización LU.
MG	Fortran	Resolución de una ecuación discreta de Poisson tridimensional con el método *V-cycle Multi-grid*.
SP	Fortran	Resolución de un sistema de ecuaciones de derivadas parciales mediante el algoritmo *Beam and Warning*.
UA	Fortran	Resolución de la transferencia de calor en un dominio cúbico.

La simulación de las aplicaciones incluidas en estas *suites* no tiene lugar en un punto arbitrario, sino que se restringe a su región de interés (*Region Of Interest* o ROI). La ROI de las aplicaciones es una porción de éstas que, generalmente, excluye fases que resultan poco relevantes para la evaluación, tales como la inicialización o la finalización (inicialización/destrucción de variables y lectura/escritura de ficheros). La ROI se presenta normalmente como un bucle principal o una secuencia de llamadas a funciones con alta carga computacional, acumulando en ambos casos la mayor parte del tiempo de ejecución de la aplicación (en el caso de PARSEC, la ROI acumula más del 80% del tiempo de ejecución en la mayoría de aplicaciones para el tamaño de problema *native* [139]). La ROI de las aplicaciones de PARSEC y NPB únicamente comprende la parte paralela del código, mientras que en las aplicaciones pertenecientes a SPEC delimita una porción del código secuencial. La limitación de las simulaciones a la ROI requiere la instrumentalización del código (tal y como se detalla en la siguiente sección del presente capítulo), una tarea que no es necesaria en el caso de PARSEC (puesto que el código fuente ya incluye la delimitación de la ROI para cada aplicación), pero que sí lo es para NPB y SPEC. Por tanto, se realiza el esfuerzo de inspeccionar el código fuente de cada aplicación (un total de 35 aplicaciones) pertenecientes a estas *suites* para proceder a identificar la ROI correspondiente y a realizar las modificaciones pertinentes en el código.

La simulación restringida a la extensión de la ROI supone un desafío adicional en el caso de la *suite* SPEC debido a la naturaleza secuencial de sus aplicaciones, puesto que, en presencia de múltiples *cores*, se opta por simular varias copias de la misma aplicación (una por *core*), sin que haya ningún tipo de sincronización entre ellas. Por lo tanto, el desfase que se produce en la ejecución de las distintas copias implica que no hay certeza de que todas lleguen a inicio de la ROI al mismo tiempo, siendo necesario la utilización de algún mecanismo de sincronización. Para tal fin, se ha diseñado un programa (en código C) que se encarga de crear distintos procesos para el lanzamiento de las múltiples copias de la misma aplicación y que les permite sincronizarse entre sí haciendo uso de una barrera de la librería Pthreads "situada" al comienzo de la ROI. La utilización de este programa garantiza que todas las copias han alcanzado un determinado punto de la aplicación, con

independencia de que se produzca un mayor o menor desfase en los diferentes flujos de ejecución, de modo que posibilita la correcta simulación de estas aplicaciones en particular.

3.2.2 Metodología de Simulación

En esta sección se resumen los principales aspectos de la metodología de simulación sobre la que se sustenta gran parte del desarrollo de la tesis. La herramienta seleccionada, gem5, es un simulador de sistema completo con un nivel de detalle suficiente para posibilitar la ejecución un sistema operativo real (sin modificaciones) y ejecutar sobre éste las aplicaciones pertinentes (ver Figura 19). Este modo de simulación de la herramienta necesita dos elementos para funcionar: un *kernel* y una imagen de disco (básicamente un sistema de ficheros). Se utiliza la versión 3.18.34 del *kernel* de Linux y una imagen de disco por cada una de las *suites* de *benchmarks* tradicionales descritas en la sección anterior. Estas imágenes están creadas tomando como base la distribución de Linux *Debian Jessie* (versión 8), añadiendo los binarios de las aplicaciones y las librerías que éstas necesitan, además del *scripting* necesario para configurar y lanzar la ejecución de las aplicaciones.

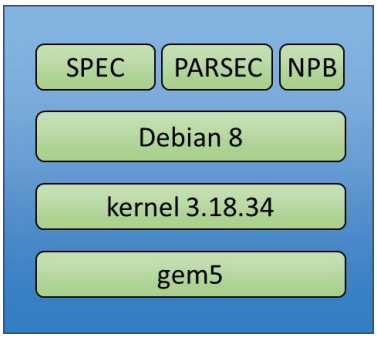

Figura 19 – Esquema del entorno de simulación utilizado.

La metodología, que se detallará más adelante, se implementa en base al uso de los distintos modelos de CPU que incluye la herramienta y que funcionan con el modo de simulación *full-system*. Cada uno de estos modelos se sitúa en un punto distinto en lo referente al *trade-off* precisión-tiempo, lo que determina su rol dentro de la metodología. A continuación, se describen brevemente las características más importantes de los tres modelos de CPU y su rol dentro de la metodología: *atomic*, *detailed* y *kvm*.

- **Atomic** es el modelo de CPU más sencillo. Modela un *core* que ejecuta una instrucción por ciclo. Es posible utilizarlo junto a una jerarquía de memoria con distintos niveles o en solitario, pero no modela el *timing* de los accesos. Este modelo resulta adecuado en aquellas situaciones donde el *timing* del *pipeline* no tiene importancia, como en las fases de calentamiento (el uso que tiene dentro de la metodología utilizada) o al realizar medidas exploratorias de *miss rate* sobre la jerarquía de memoria.

- **Detailed** es el modelo de CPU más complejo implementado en la herramienta. Modela un *pipeline* superescalar, con ejecución fuera de orden, basado en la arquitectura del Alpha 21264. Este es el modelo que se utiliza para llevar a cabo la simulación detallada de la micro-arquitectura.

- **Kvm** (*Kernel-based Virtual Machine*) es un modelo que hace uso de simulación asistida por virtualización hardware [106], liberando así a la herramienta del modelado de la funcionalidad de las instrucciones y del *timing*. El aspecto más destacable de este modelo

es la velocidad de simulación que ofrece (casi nativa), haciéndolo adecuado para la tarea de preparación de las cargas de trabajo de las aplicaciones.

La metodología empleada está representada en la Figura 20. Utiliza los tres modelos de CPU brevemente descritos y consta de tres fases donde se hace uso de los distintos modelos (uno por fase): la inicialización, el *warmup* y la simulación detallada de la micro-arquitectura. En lo que resta de sección se detallan los aspectos más relevantes de esta metodología.

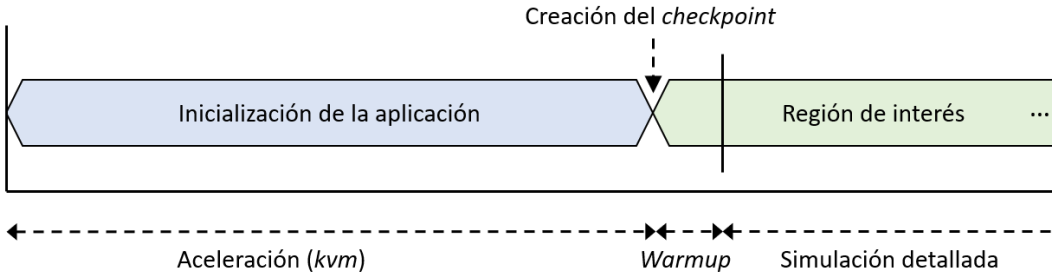

Figura 20 – Resumen de la metodología.

El primer paso de la metodología consiste en alcanzar la ROI de las aplicaciones. Esta tarea es muy costosa (computacionalmente) debido a que el número de instrucciones que hay que simular es, en general, muy elevado. Por este motivo, se hace uso del modelo de CPU *kvm* de gem5, logrando de esta manera reducir el proceso de inicialización a su mínima expresión. *Kvm* es el único modelo de CPU de la herramienta capaz de alcanzar la ROI en un tiempo asumible, puesto que con el siguiente modelo más rápido (*atomic*) esta tarea se alargaría fácilmente durante varios meses en algún caso. El siguiente paso consiste en crear un *checkpoint* en el momento en el que se alcanza la ROI. Su creación, en un punto concreto de la aplicación, es posible gracias a que el simulador es capaz de reconocer determinadas instrucciones especiales, denominadas *m5 ops*, de tal manera que, mediante la modificación del código fuente de las aplicaciones, es posible realizar determinadas acciones como interrumpir la simulación, reiniciar las estadísticas, o, en este caso, crear un *checkpoint*. Gracias a la creación de *checkpoints*, la tarea de alcanzar la ROI solamente debe completarse una única vez, puesto que el simulador es capaz de restaurar la simulación desde un punto concreto cuantas veces sean necesarias.

Los trabajos de simulación continúan a partir del estado salvado en los *checkpoints* y constan de dos fases, comienzan con una fase de *warmup* para, a continuación, pasar a simular, con gran detalle, la micro-arquitectura. Al restaurar la simulación desde un *checkpoint*, los distintos niveles de la jerarquía de memoria *on-chip* se encuentran vacíos de contenido y, por tanto, su tasa de fallos es muy alta hasta que adquieren contenido y comienzan a funcionar con normalidad. Por este motivo, todos los trabajos comienzan realizando un *warmup* de extensión configurable (usualmente unos pocos cientos de millones de instrucciones) con el objetivo evitar que las medidas se vean afectadas por el mal funcionamiento inicial de la jerarquía de memoria. Para minimizar el impacto de esta fase en la duración de los trabajos se hace uso del modelo *atomic*, dado que el único objetivo es el calentamiento de la jerarquía de memoria. Una vez que finaliza el *warmup*, se reinician las estadísticas que recoge la herramienta y se hace uso de una característica muy importante de la herramienta, *el "*cambio en vuelo" del modelo de CPU. A través de los *scripts* que crean y gobiernan la simulación, es posible migrar de un modelo de CPU a otro, de modo que, en este caso particular, se abandona el modelo *atomic* por el *detailed* al finalizar el *warmup* para comenzar con la simulación detallada. La extensión de esta segunda fase de los trabajos de simulación puede configurarse de dos modos distintos, estableciendo un límite de instrucciones o de transacciones. En el primer caso, la simulación se interrumpe cuando uno

de los *cores* del sistema alcanza un determinado número de instrucciones ejecutadas, mientras que, en el segundo caso, se interrumpe cuando la simulación ejecuta un número determinado de transacciones (cada transacción implica una iteración en el bucle principal de la aplicación).

Cada simulación se lleva a cabo múltiples veces hasta alcanzar un intervalo de confianza del 95%. La simulación de las aplicaciones no es determinista, sino que la herramienta genera cierta variabilidad en base a incrementar ligeramente la latencia de acceso a memoria principal. Estas pequeñas fluctuaciones provocan que los accesos se reordenen y son capaces de generar cierta variabilidad en los resultados de las simulaciones.

3.3 Complejidad vs Precisión: Adaptando el *Framework* de Trabajo a Aplicaciones Emergentes

Las aplicaciones tradicionales (sección 3.2.1) utilizadas en el área de Arquitectura de Computadores son cada vez menos representativas de entornos actuales [27]. Aplicaciones emergentes pertenecientes a entornos de *Cloud Computing,* tales como el almacenamiento y procesado distribuido tienen cada vez más cuota de mercado, lo que fuerza a que sean incluidas como parte de la evaluación en este campo. Por este motivo, una parte del trabajo que se ha llevado a cabo en esta tesis se ha dedicado a mejorar la representatividad de la evaluación mediante la ampliación de *benchmarks* compuestos por aplicaciones emergentes, en concreto con bases de datos NoSQL. La adopción de YCSB, un *framework* que permite evaluar el rendimiento de múltiples bases de datos NoSQL, motiva la adaptación de la metodología. En lo que resta del presente capítulo se detalla el proceso que se ha llevado a cabo para adoptar la metodología utilizada a las nuevas aplicaciones. Adicionalmente, se demuestran los posibles problemas derivados de la utilización de metodologías no adecuadas, que pueden implicar pérdidas de precisión en las medidas que lleven a conclusiones completamente erróneas.

3.3.1 *Benchmarking* de Aplicaciones NoSQL (YCSB)

Las bases de datos NoSQL son aplicaciones radicalmente distintas a aquellas que forman parte de *benchmarks* más convencionales en el área, como SPEC o PARSEC. Trabajan con una arquitectura software cliente-servidor, lo que implica que el flujo de instrucciones (comportamiento) ejecutado por el servidor (aplicación a evaluar) es condicionado por la interacción que los distintos clientes hacen sobre el mismo (siendo dicha interacción en este caso el almacenamiento de datos y la ejecución de peticiones sobre dichos datos). Adicionalmente, la generación de contenidos en la base de datos sobre los que interactúan los clientes requiere un costoso proceso de inicialización previo a la evaluación de su rendimiento, correspondiente a la creación y el almacenamiento de los datos. Dado que el rol de las bases de datos a analizar se limita a la parte servidor, es necesario incorporar un cliente cuyo rol será la creación de la estructura de datos a almacenar, así como la posterior generación de distintos tipos de peticiones para evaluar el rendimiento de las aplicaciones NoSQL. El cliente que se va a utilizar es YCSB (*Yahoo Cloud Benchmark Suite*) [137], un *framework* que posibilita la evaluación de rendimiento de un amplio conjunto de bases de datos NoSQL (más de 30 están incluidas en su código fuente). Gracias a dicha versatilidad, ofrece un entorno de trabajo uniforme para la evaluación de las cuatro bases de datos descritas en las secciones 2.4.1-2.4.4 (Redis, MongoDB, OrientDB y Cassandra), pues YCSB implementa una interfaz para todas ellas.

La organización interna de los datos creada por YCSB consiste en una tabla de *registros*, los cuales se subdividen en un determinado número de campos. Dicha tabla es altamente configurable, a

través parámetros como el número de campos por registro o el tamaño de cada campo. De esta forma se obtiene control completo sobre el tamaño y la organización de la base de datos. El contenido almacenado en cada campo es una secuencia de caracteres ASCII generada de manera aleatoria y, por defecto, cada registro contiene un total de 10 campos y cada campo tiene un tamaño de 100 bytes. El acceso y manipulación de los datos se lleva a cabo a nivel de registro, siendo posible la configuración de su extensión en base al número de campos del registro involucrados en la operación. Así, las operaciones pueden afectar a un único campo, seleccionado de manera aleatoria, o a la totalidad de los campos que componen el registro. La configuración por defecto establece que las operaciones de lectura afectan a todos los campos, mientras que las escritura se limitan a uno solo. El conjunto de operaciones implementado corresponde a las consultas (*queries*) habituales en cualquier base de datos, inserción de un nuevo registro (*insert*), lectura (*read*), modificación (*write*), lectura+modificación (*read-modify-write*) del contenido de un registro y lectura secuencial de múltiples registros (*scan*). Finalmente, el patrón de acceso a los datos almacenados también es configurable de acuerdo con las distintas funciones de probabilidad implementadas: *uniform*, *zipfian*, *latest*, *sequential*, *exponential* y *hotspot*.

Tabla 5 – Resumen de la configuración de YCSB [137].

Workload	**Description**	**Operations**	**Distribution**
A	Update heavy (Session store recording recent actions in a user session)	50% reads, 50% writes.	Zipfian
B	Read mostly (Photo tagging; add a tag is an update, but most operations are to read tags)	95% reads, 5% writes.	Zipfian
C	Read only (User profile cache, where profiles are constructed elsewhere)	100% read.	Zipfian
D	Read latest (User status update, people want to read the latest status)	95% read, 5% insert.	Latest
E	Short ranges (threaded conversations, where each scan is for the posts in a given thread)	95% scan, 5% insert.	Zipfian/Uniform
F	Read-Modify-write (user database, where user records are read aun modified by the user)	50% read, 50% r-m-w	Zipfian
Number of fields = 10 / Bytes per field = 100B / Reads = All fields / Writes = One field			

En base al conjunto de consultas y patrones de acceso definidos, YCSB implementa seis *workloads* que pretenden representar escenarios de ejecución habituales en entornos en los que se utilizan este tipo de bases de datos (ver Tabla 5). En dichos *workloads* se define la fracción de consultas de cada tipo realizadas, así como la distribución de dichos accesos a los datos. Además del conjunto de *workloads* predefinidos, YCSB incluye una plantilla para la creación de *workloads* personalizados.

La ejecución de las cargas de trabajo definidas en YCSB consta de dos partes bien diferenciadas. En primer lugar, es necesario llevar a cabo la carga de la base de datos, proceso que se realiza de acuerdo con los parámetros que definen la estructura de los datos, a través de operaciones de tipo *update*. En segundo lugar, se ejecuta un número de consultas predefinido sobre la base de datos creada, atendiendo al patrón de datos existente en la definición de cada *workload*.

3.3.2 Evaluación de Entornos Distribuidos: gem5+YCSB+NoSQL

La adopción del *framework* YCSB y las bases de datos NoSQL, como parte del banco de pruebas, motiva un proceso de adaptación de la herramienta de trabajo debido a la naturaleza distribuida de estas aplicaciones, así como a sus requerimientos de ejecución (generación previa del *dataset*).

La arquitectura distribuida de las bases de datos NoSQL plantea la necesidad de poder llevar a cabo simulaciones multi-nodo a semejanza de entornos reales. Sin embargo, el enfoque *single-node* de la mayoría de las herramientas provoca que la simulación de este tipo de aplicaciones suponga un reto para la metodología. A fecha de redacción de la presente tesis, la simulación multi-nodo es una característica disponible en la rama principal de gem5, gracias a soluciones como la propuesta en [140] y denominada *dist-gem5*, donde la simulación de múltiples nodos se ejecuta de forma distribuida en múltiples equipos físicos, utilizando un canal de comunicación TCP (*Transmission Control Protocol*) para intercambiar mensajes de datos y sincronización entre los nodos que componen la simulación. Desafortunadamente, *dist-gem5* es posterior al trabajo realizado en esta tesis, y en su momento fue necesario llevar a cabo las reparaciones pertinentes para poder realizar simulaciones multi-nodo. Para lograr ésto, fue necesario modificar el código correspondiente a la construcción del sistema, que solamente permitía simulaciones *single-node* o *dual-node* con conexión punto a punto. El trabajo con la herramienta permitió la simulación de sistemas multi-nodo comunicados a través de un *switch ethernet* también simulado. Con el conjunto de modificaciones realizadas se logró adecuar la herramienta de simulación a la arquitectura de estas aplicaciones, pudiendo simular múltiples sistemas con el nivel de precisión requerido.

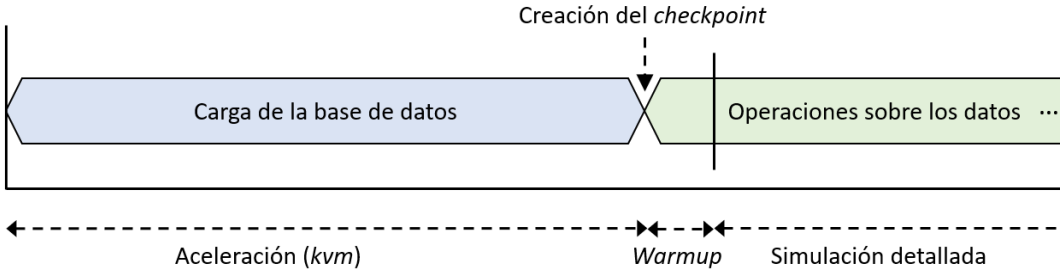

Figura 21 – Proceso completo de la creación de los workloads y de la simulación.

El procedimiento de preparación de las cargas de trabajo para las aplicaciones NoSQL es similar al de las aplicaciones tradicionales (Sección 3.2.2), utilizando simulación asistida por hardware (*kvm*) para llevar la ejecución de la aplicación pertinente hasta la ROI, creando en dicho punto un *checkpoint* desde el cual poder restaurar la simulación (ver Figura 21) y realizar medidas con alto nivel de detalle en el modelado de los componentes hardware. En este caso, se define la región de interés para este tipo aplicaciones como el punto en el que la base de datos ha sido cargada por completo, estando en disposición de comenzar la ejecución de los distintos *workloads* definidos por YCSB (sección 3.3.1). La simulación del proceso de carga presenta un coste computacional elevado, pues se trabaja con bases de datos de tamaño realista (del orden de 1GB) que requieren gran cantidad de inserciones para generarse. La sección 3.3.4 demostrará que la utilización de aceleración hardware (*kvm*) es indispensable para conseguir tiempos, de carga de la base de datos, asumibles. El correcto funcionamiento en gem5 del modelo de CPU basado en *kvm* ha supuesto un problema persistente durante el desarrollo de esta tesis. Los cambios introducidos en el código fuente (por parte de la comunidad de usuarios) referente a *kvm*, con el objetivo de hacerlo funcionar con las particularidades de los procesadores AMD e Intel, provocaron, en contra de sus intenciones, que *kvm* únicamente fuese capaz de funcionar con AMD, siendo necesaria la revisión del código fuente para lograr que volviese a funcionar con Intel apropiadamente (el hardware disponible para el desarrollo de esta tesis). Adicionalmente, ha sido necesario trabajar sobre los *scripts* de construcción del sistema para conseguir que *kvm* haga uso de las colas de eventos *multithread* implementadas en gem5, logrando una implementación de *kvm multithread*. De esta manera, se ha logrado que los trabajos de simulación que utilizan el modelo de CPU

basado en *kvm* se ejecuten a velocidad casi nativa (siempre que hubiera tantos procesadores físicos como procesadores simulados), posibilitando el avance, hasta la región de interés de las aplicaciones a velocidad, prácticamente, nativa.

Las peculiaridades de este tipo de aplicaciones se centran en la fase de creación de las cargas de trabajo, siguiendo la fase de simulación detallada un proceso similar al de las aplicaciones tradicionales, comenzando con una fase de *warmup* donde se utiliza la descripción de hardware más sencilla posible (*atomic*) para, a continuación, pasar a simular con el modelo más detallado (*detailed*).

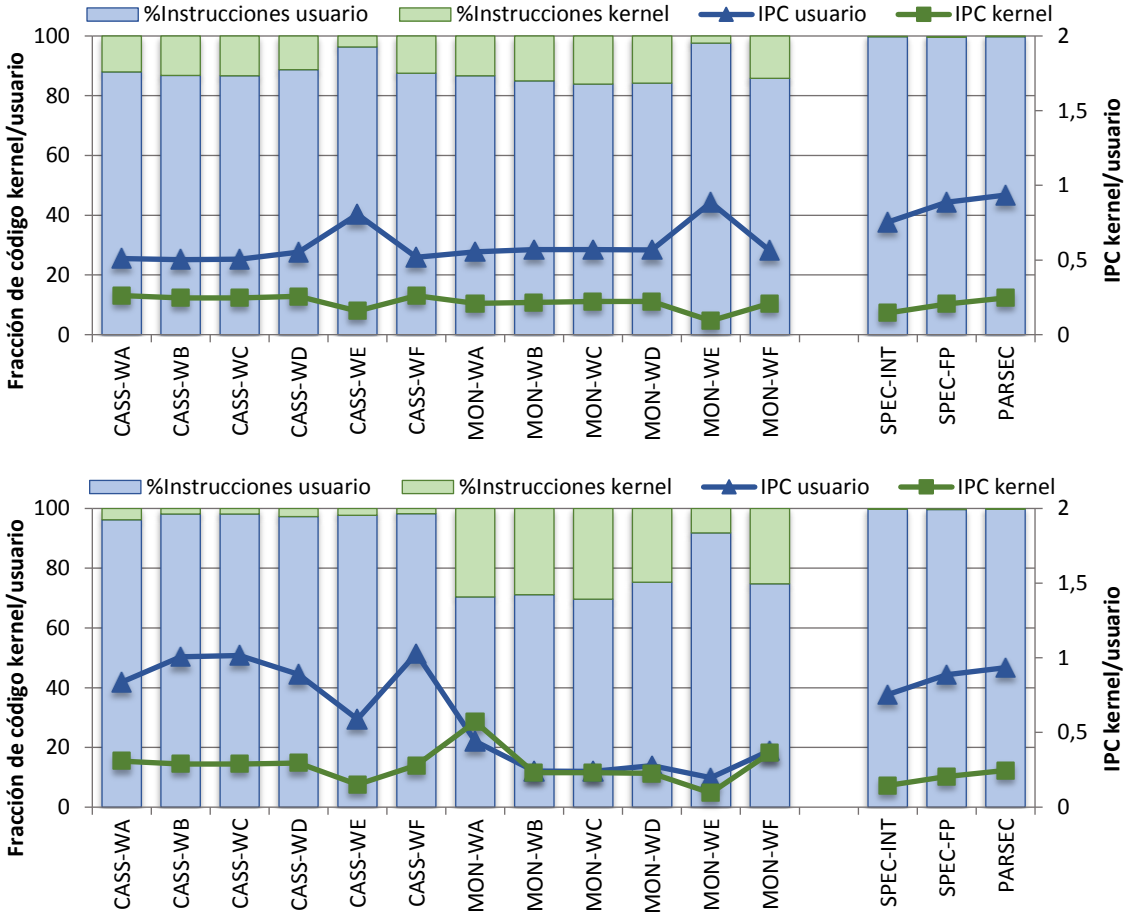

Figura 22 - Porcentaje de instrucciones e IPC en modo kernel y usuario. (arriba) Resultados del cliente (YCSB). (abajo) Resultados de las bases de datos Cassandra y MongoDB.

Un último aspecto destacable para la apropiada simulación de las aplicaciones NoSQL es la importancia del sistema operativo. Se ha comprobado este hecho midiendo la fracción de código ejecutado (instrucciones retiradas) en modo usuario y supervisor en la ejecución de cargas de trabajo NoSQL (*workloads* de YCSB sobre Cassandra y MongoDB) y de dos *benchmarks* tradicionales (SPEC y PARSEC). Las ejecuciones de este experimento se realizan sobre hardware real, utilizando las PMU de los procesadores (a través de la herramienta *perf* [98]) para monitorizar y medir los datos deseados. Las bases de datos NoSQL presentan un *dataset* de 12GB de tamaño, monitorizando la ejecución de 1 millón de operaciones sobre Cassandra y 2 millones sobre MongoDB. En el caso de SPEC y PARSEC se monitoriza la ejecución completa de las aplicaciones. Los resultados del experimento se presentan en la Figura 22, cuyo eje vertical muestra la fracción (columnas) e IPC (*Instructions Per Cycle*) (línea) de instrucciones de usuario y supervisor. Se puede observar que la fracción de código supervisor en el caso de PARSEC y SPEC es prácticamente nula

y, por tanto, la influencia de su emulación en los resultados no parece crítica. Sin embargo, este hecho no es extensible a las aplicaciones emergentes, pues tanto Cassandra como MongoDB presentan una fracción de código privilegiado nada despreciable, al igual que el cliente de YCSB. En lo que concierne este último, se observan resultados simulares al interactuar tanto con Cassandra como con MongoDB, con un porcentaje de instrucciones de sistema operativo que se sitúa en la mayoría de los casos por encima del 10% y superando el 15% en algún caso particular. Con respecto a las bases de datos, Cassandra presenta una fracción de instrucciones de sistema operativo cercana al 5%, alcanzando MongoDB el 30% en la ejecución de varios *workloads*. Por otra parte, se observan sustanciales diferencias en la mayor parte de las medidas entre el IPC de instrucciones en modo usuario y el IPC de instrucciones en modo privilegiado (con la excepción de MongoDB), lo que sugiere que el código ejecutado en ambos modos es bien distinto y motiva aún más la necesidad de incluir al sistema operativo en las simulaciones. Los resultados de este experimento corroboran las observaciones realizadas por [30], y confirman las diferencias existentes entre las aplicaciones tradicionales y las aplicaciones emergentes en lo referente al sistema operativo. Una simplificación habitual en algunas herramientas de simulación [55] es la no simulación del sistema operativo, recurriendo a herramientas basadas en trazas que simulan únicamente el código de las aplicaciones o a herramientas que emulan el comportamiento de las "llamadas al sistema". Los efectos de esta simplificación son despreciables para aquellas aplicaciones que ejecuten un porcentaje mínimo de instrucciones privilegiadas, pero no resulta adecuada para las bases de datos NoSQL puesto que ejecutan mucho código de sistema operativo [30]. Afortunadamente, gem5 implementa un modo de simulación con la precisión necesaria en la implementación del repertorio de instrucciones (ISA) para ejecutar un sistema operativo sin modificaciones.

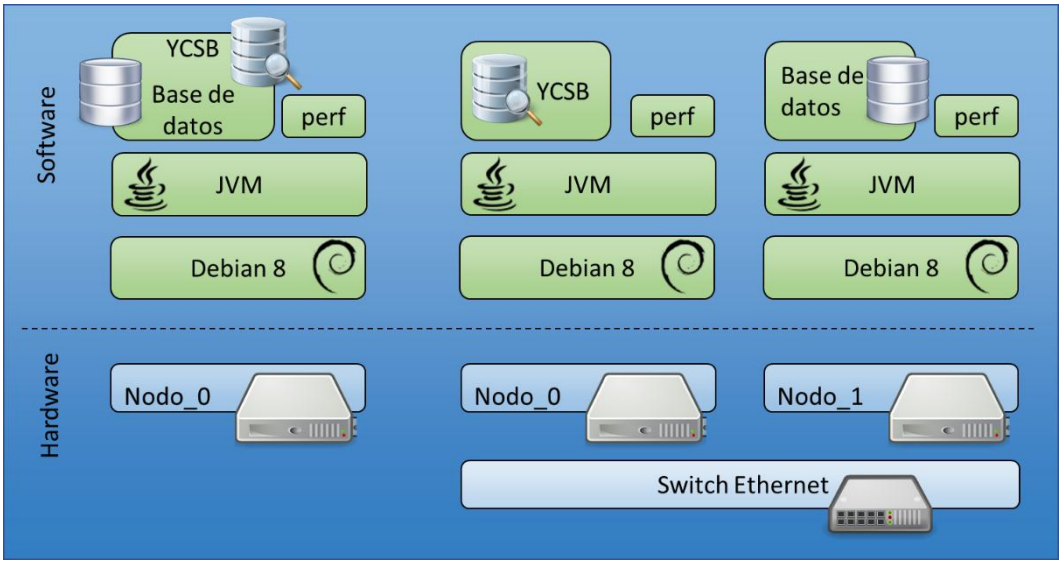

Figura 23 – Esquema de la monitorización hardware con dos nodos. (izquierda) 1 nodo. (derecha) 2 nodos.

3.3.3 Simulaciones Multi-nodo: Precisión vs. Coste Computacional

Una vez descritas las peculiaridades de las aplicaciones NoSQL con respecto a la metodología de simulación, es necesario determinar si la simulación de estas aplicaciones en múltiples nodos es un requerimiento estricto o si, por el contrario, la simplificación de simular el cliente y la base de datos en un único nodo no tiene efectos significativos en los resultados. La complejidad asociada a la simulación de sistemas multi-nodo, en términos de tiempo y memoria (sección 3.3.4), hace que resulte tentador simular aplicaciones distribuidas en un único sistema. Sin embargo, la

interacción de las diferentes partes del software (cliente y servidor) es una posible fuente de error que podría distorsionar las medidas de rendimiento de una hipotética propuesta micro-arquitectural. Por este motivo, en esta sección se pretende cuantificar el error causado por tal interacción, a través de la realización de múltiples experimentos comparando la ejecución de las aplicaciones distribuidas NoSQL en un único nodo y en sistemas multi-nodo. En concreto, se evalúa la tasa promedio de fallos sobre el último nivel de *cache* (LLC o *Last Level Cache*), la *cache* de primer nivel de instrucciones y en el TLB (*translation lookaside buffer*) de datos.

Para llevar a cabo el proceso de evaluación descrito se utilizarán dos metodologías distintas y complementarias. Por un lado, una parte de los experimentos se van a llevar a cabo sobre hardware real, monitorizando la ejecución de las aplicaciones NoSQL mediante el uso de los contadores hardware de la PMU (sección 3.1.1) de los nodos disponibles. El hardware en estas pruebas consiste en procesadores Intel Xeon S5650 (12 *cores* y un total de 24 *threads* a 2,67Ghz) con 48GB de memoria principal por nodo. La Figura 23 esquematiza la metodología descrita. Se mantiene la configuración de base de datos de la sección 3.3.2 (12GB de tamaño, 1 millón y 2 millones de operaciones por *workload* para Cassandra y MongoDB respectivamente). Por otro lado, se utilizará la herramienta de simulación gem5 en aquellos experimentos que requieran evaluar cambios en la micro-arquitectura del procesador. En este caso, con el fin de trabajar con tiempos de simulación asumibles, será necesario llevar a cabo un escalado del tamaño del problema. Se reduce el tamaño de la base de datos a 1GB, el número de operaciones a 1.000 y se escala el número de *cores* del procesador de 12 a 4. Es necesario destacar en este punto, como se comprobará en el siguiente capítulo, que el escalado de parámetros propuesto para simulación no tiene efectos apreciables en el comportamiento observado en la jerarquía de *cache* del procesador.

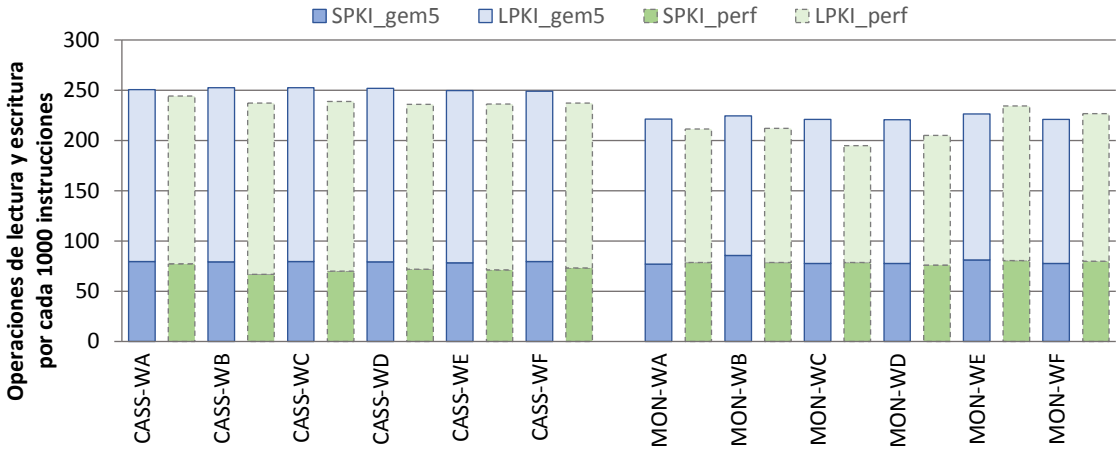

Figura 24 – Fracción de instrucciones de lectura y escritura en memoria por cada mil instrucciones ejecutadas comparando dos entornos de ejecución (gem5 (azul) y hardware real (verde)).

Dada la utilización conjunta de diferentes metodologías de evaluación, se antoja conveniente la validación de la utilización simultánea de ambas. Para ello se ha llevado a cabo un experimento preliminar, consistente en la utilización de YCSB para crear una base de datos de idéntico tamaño en cada entorno y medir la fracción de operaciones de acceso a memoria (*loads*/*stores*) para la ejecución de un mismo *workload*. Se utilizan las bases de datos Cassandra y MongoDB, con un tamaño de 1GB para los datos, y se obtienen los resultados de la ejecución de 1.000 operaciones utilizando los distintos *workloads* que incluye YCSB. La Figura 24 muestra el número de lecturas y escrituras por cada mil instrucciones ejecutadas en ambos entornos, donde las barras azules representan los resultados obtenidos con gem5 y las verdes los del hardware real. Estos resultados

indican que la desviación de gem5 sobre el hardware real es mínima, no llega al 5% en media, lo que valida la utilización conjunta de ambas metodologías. Dicha desviación entra dentro de los márgenes razonables, teniendo en cuenta que gem5 no hace una traducción fidedigna de operaciones de alto nivel a micro-instrucciones (dichas micro-instrucciones son diferentes para cada fabricante y no están disponibles públicamente).

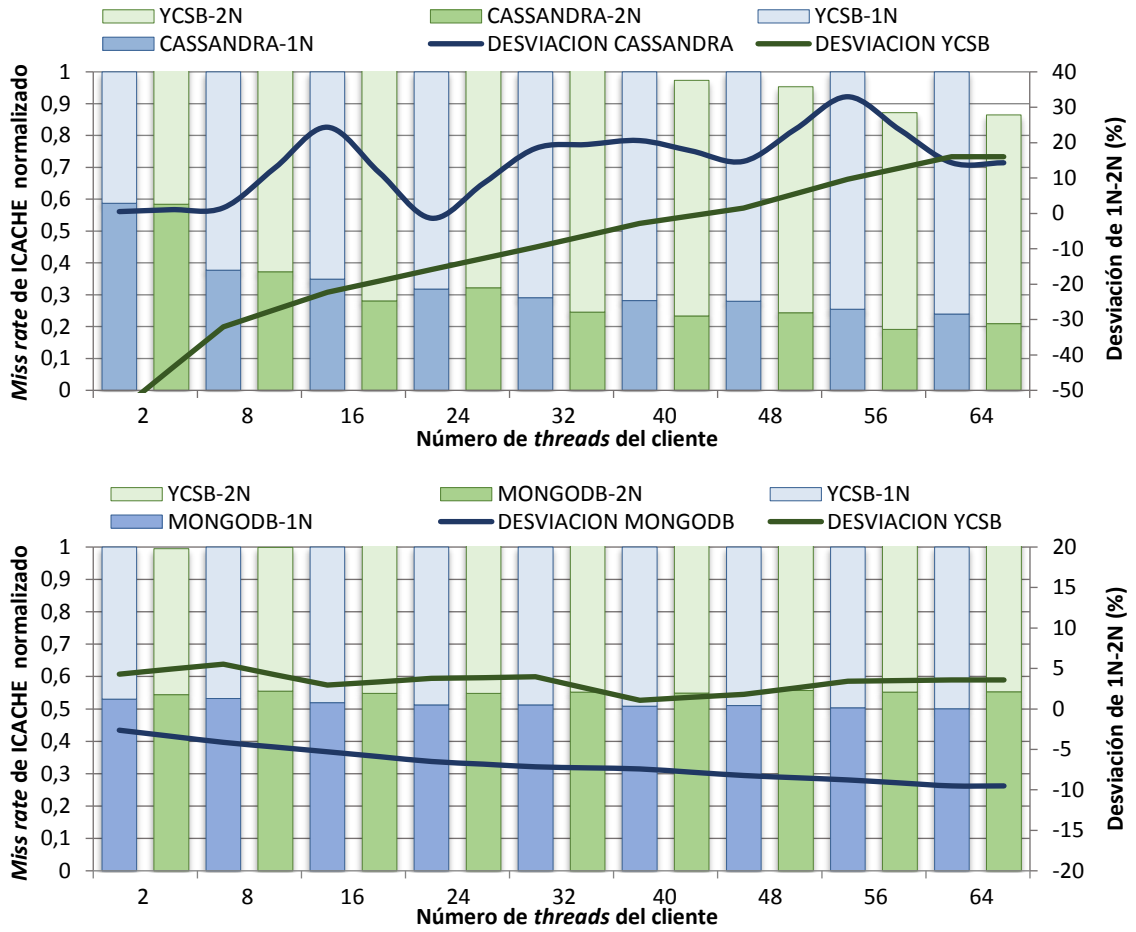

Figura 25 – Evolución del miss rate en ICACHE (normalizado a los valores de 1 nodo) a medida que aumenta el número de threads del cliente. (arriba) Cassandra. (abajo) MongoDB. Valores de miss rate (barras) y desviación (líneas).

Tras el proceso de validación, se comienza la evaluación analizando los efectos de la interacción del cliente y servidor en estructuras hardware privadas como son la *cache* de instrucciones y el TLB de datos utilizando los contadores hardware. La monitorización incluye tanto al cliente como al servidor (la base de datos). Para este primer experimento se limitan los resultados mostrados a un único *workload* de YCSB (*Workload* A), ejecutado para dos bases de datos distintas (Cassandra y MongoDB). El resto de *workloads* de YCSB también han sido evaluados, presentando unos resultados similares a los mostrados aquí. La Figura 25 muestra la tasa de fallos (o *miss rate*) de la *cache* de instrucciones cuando el cliente y la base de datos se ejecutan en el mismo sistema (1N) y cuando se ejecutan en dos sistemas (2N). Los resultados de las medidas muestran la evolución de la tasa de fallos, tanto para el cliente como para la base de datos, a medida que aumenta el número de *threads* del cliente (YCSB), en un rango de 2 a 64. La figura muestra los valores individuales de la tasa de fallos normalizados a los valores obtenidos con 1 nodo (barras). Además, muestra las líneas de desviación tanto para el cliente como para la base de datos, es decir, el error que se produce en cada medida al ejecutar el cliente y el servidor en un único sistema frente a su ejecución en sendos sistemas. Nótese que la escala del eje Y secundario no es idéntica en ambas

gráficas. Cuando el número de fallos en 1N es menor que en 2N la desviación es negativa, mientras que es positiva cuando es mayor.

Los resultados obtenidos de la monitorización de Cassandra revelan que la desviación no es despreciable, ni para el cliente ni para la base de datos. En el caso del cliente, el error evoluciona de valores negativos elevados cuando el número de *threads* es pequeño a valores positivos más pequeños cuando este número se incrementa, situándose en el peor caso por encima del 50%. Los resultados de la desviación del servidor son más consistentes, cuyo valor es positivo en la mayoría de las medidas y se sitúa en el 30% en el peor de los casos. En lo concerniente a MongoDB, la desviación es más contenida y mucho más homogénea entre las distintas medidas realizadas. Los valores se sitúan por debajo del 5% para el cliente y crecen ligeramente hasta al 10% en el caso del servidor. Es previsible que la fuente de error en estas medidas esté relacionada con la diferencia de tamaño entre el *working set* de instrucciones del cliente y del servidor. Mientras que el cliente en estos *benchmarks* (YCSB) es un simple generador sintético de peticiones implementado con un número reducido de líneas de código, las bases de datos NoSQL son bien conocidas por sus elevados *working sets* de instrucciones [27].

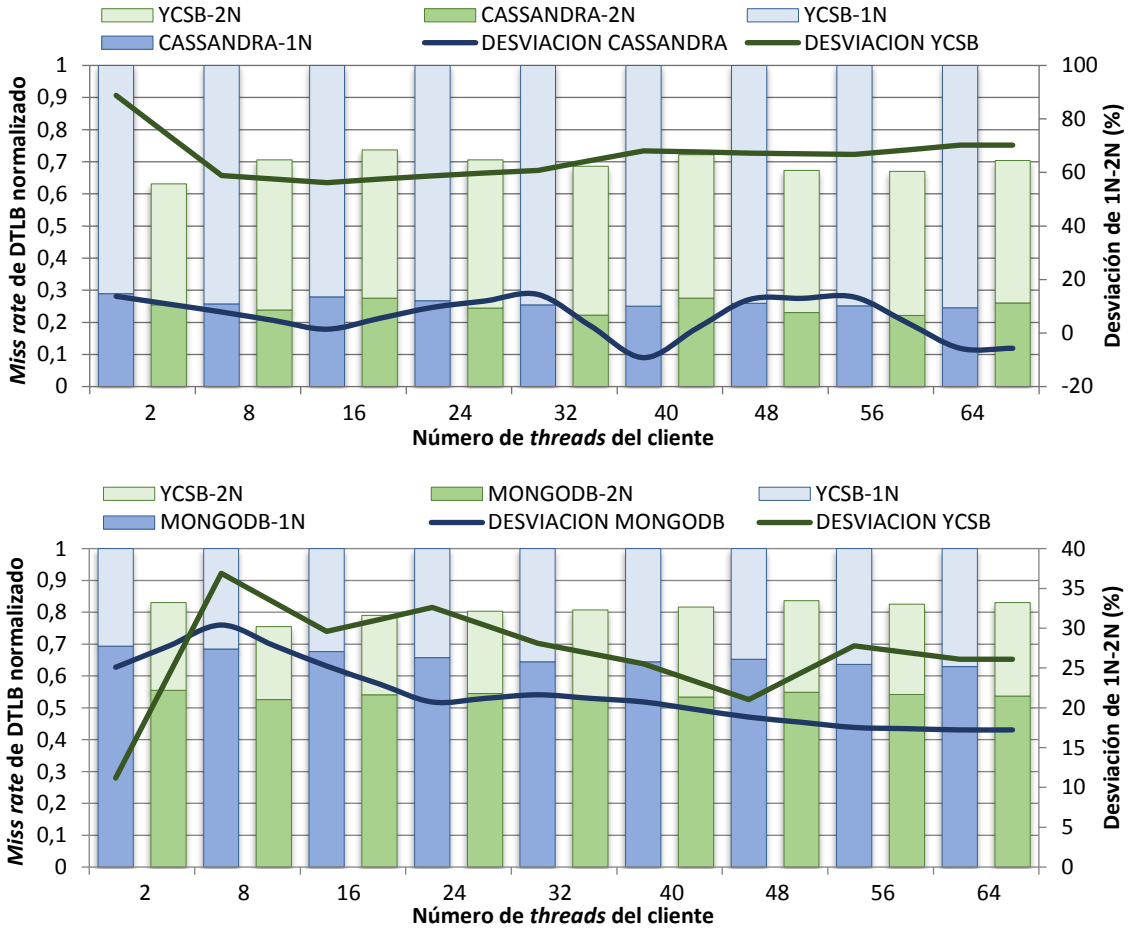

Figura 26 – Evolución del miss rate en DTLB (normalizado a los valores de 1 nodo) a medida que aumenta el número de threads del cliente. (arriba) Cassandra. (abajo) MongoDB. Valores de miss rate (barras) y desviación (líneas).

Se repite el experimento anterior, monitorizando en este caso la tasa de fallos en el TLB de datos. Los resultados se muestran en la Figura 26, donde es posible apreciar que el error que se comete, al ejecutar ambas partes en un solo nodo, crece con respecto a la *cache* de instrucciones. En el caso de Cassandra, los resultados son más homogéneos que los obtenidos en la *cache* de instrucciones, situándose la desviación, principalmente, entre el 10% y el 70%. Es mayor en el caso

del cliente, pero excede el 10% para el servidor. En el caso de MongoDB, los valores de la desviación son bastante homogéneos y se sitúan en el 30% para YCSB y en el 20% para el servidor. De la misma forma que en el caso de las instrucciones, el desequilibrio entre los tamaños de *working set* de cliente y servidor pueden estar detrás de los resultados obtenidos.

En general, los resultados obtenidos al monitorizar ambos componentes sugieren que la simplificación de ejecutar el cliente y el servidor en un único nodo provoca un error en las medidas suficiente como para conducir a conclusiones incorrectas. Sin embargo, la interacción observada es evitable a través del uso de herramientas de planificación disponibles en el sistema operativo. A cada proceso se le puede asignar una afinidad de CPU distinta (a través del comando *taskset*), evitando así la interacción cliente-servidor en aquellos componentes privados de cada CPU (como son, en este caso, la *cache* de instrucciones o el TLB de datos). Desafortunadamente, parte de los recursos del procesador son compartidos por todas las CPUs disponibles, y en este caso la interacción es inevitable. Un ejemplo claro de recurso compartido es el último nivel de *cache* (LLC), el componente de la jerarquía de memoria donde se almacenarán bloques de datos e instrucciones tanto de cliente como del servidor. Por esta razón, se amplía el análisis a este componente del procesador.

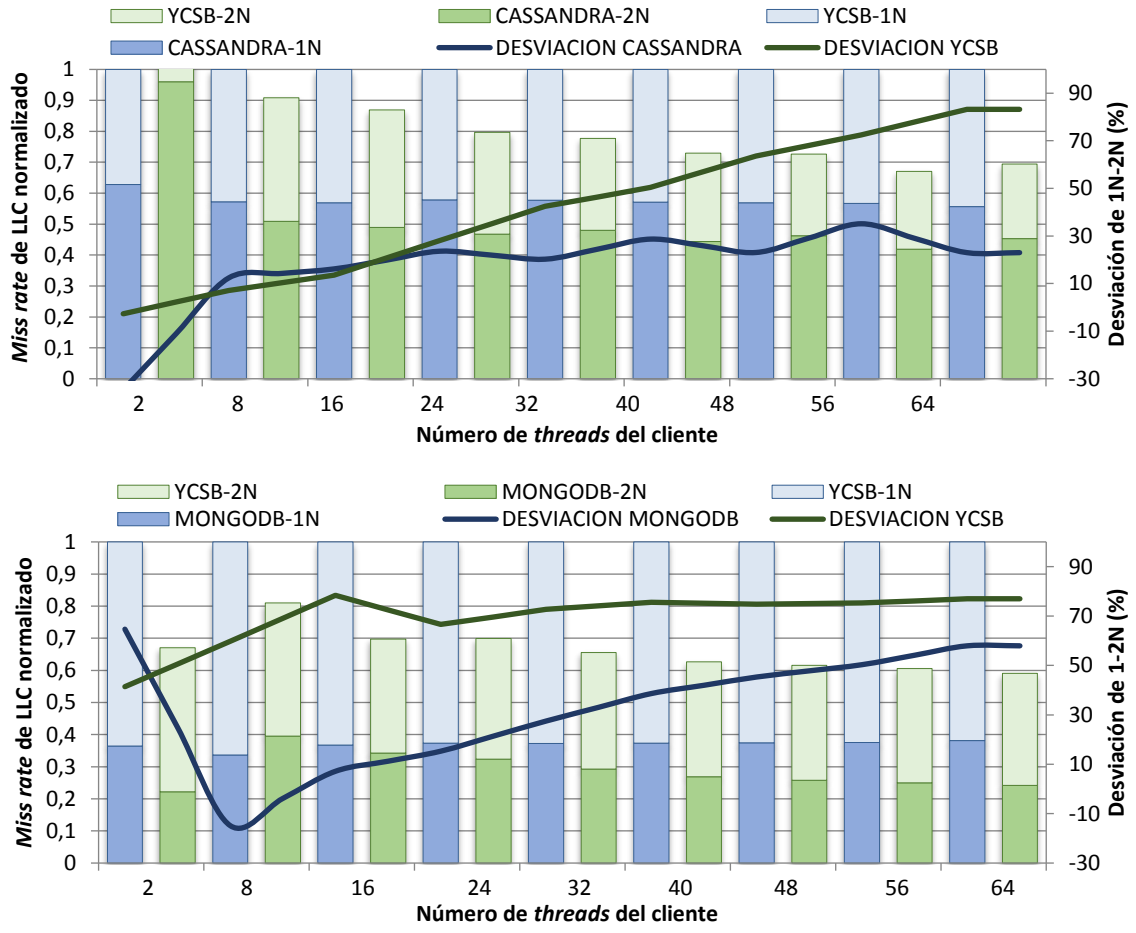

Figura 27 – Evolución del miss rate en LLC (normalizado a los valores de 1 nodo) a medida que aumenta el número de threads del cliente. (arriba) Cassandra. (abajo) MongoDB. Valores de miss rate (barras) y desviación (líneas).

La primera parte de la cuantificación del error cometido al simplificar la ejecución de aplicaciones cliente-servidor en la LLC es análoga a los dos experimentos anteriores. Los resultados que muestra la Figura 27 evidencian que los efectos de la interacción son mayores en esta estructura hardware que en las dos anteriores, sobre todo en el caso del servidor. La desviación de YCSB, en

el caso de Cassandra, muestra una tendencia alcista a medida que aumenta el número de *threads* hasta situarse por encima del 80%, mientras que en el caso del servidor se sitúa en un rango comprendido entre el 20% y el 30% en la mayoría de las medidas. En lo concerniente a MongoDB, la desviación del cliente arroja de nuevo valores muy elevados, situándose alrededor del 80% en la mayoría de las medidas. Además, la desviación de la base de datos también es muy significativa, con un valor de peor caso del 60%. La compartición de espacio por parte de datos e instrucciones, unido a las divergencias en los *working sets* de datos e instrucciones de cliente y servidor parecen tener un efecto acumulativo sobre el error cometido al evaluar los *benchmarks* NoSQL sobre un solo nodo. Dado el significativo error cuantificado en esta estructura parece razonable extender el estudio. Se abandona el uso de la PMU en favor del simulador para analizar la evolución del error a medida que crece el tamaño de la *cache*. En este experimento se continúa trabajando con las bases de datos Cassandra y MongoDB, escalando el tamaño del problema como se indicó al comienzo de la sección (Base de datos de 1GB, 1.000 peticiones por simulación) y extendiendo la evaluación a los seis *workloads* de YCSB.

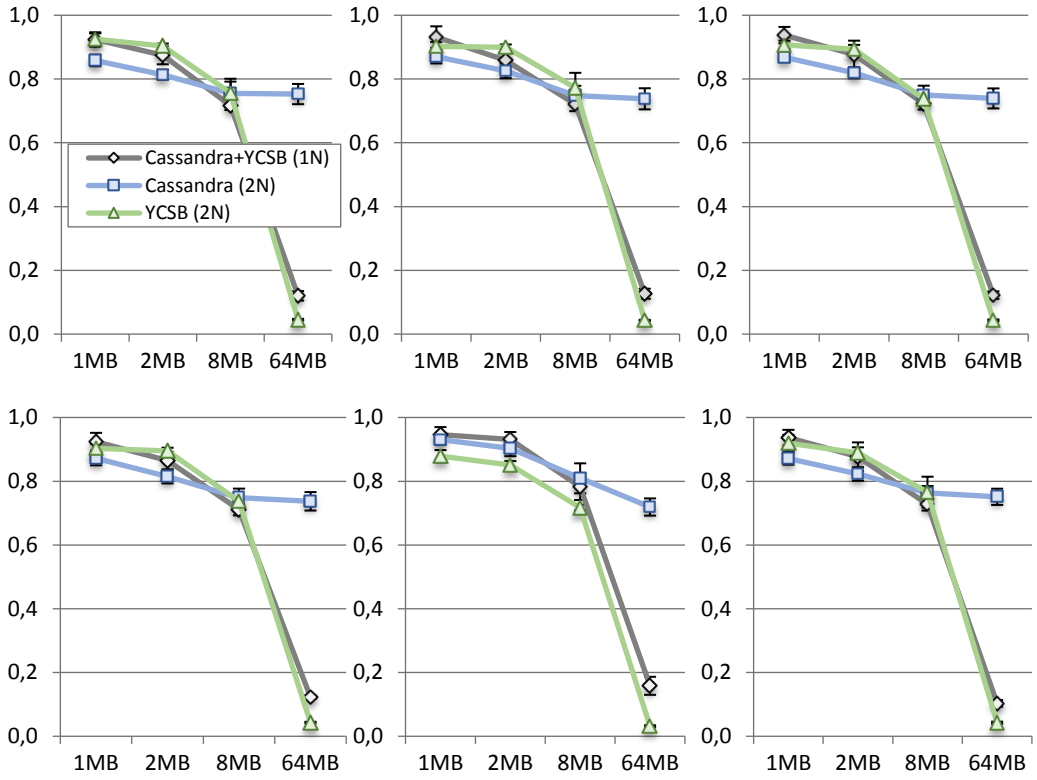

Figura 28 – Evolución del miss rate (eje Y) en LLC en función la capacidad para la base de datos Cassandra (eje X). (arriba izquierda) Workload-A. (arriba centro) Workload-B. (arriba derecha) Workload-C. (abajo izquierda) Workload-D. (abajo centro) Workload-E. (abajo derecha) Workload-F.

La Figura 28 recoge los resultados del experimento para la base de datos Cassandra. Para cada *workload*, muestra la evolución de la tasa promedio de fallos de la LLC a medida que la capacidad de este componente se incrementa desde 1MB hasta alcanzar 64MB. Los resultados obtenidos evidencian la distorsión que introduce el cliente cuando se simula junto al servidor en el mismo sistema, un efecto que se refuerza a medida que aumenta el tamaño de la *cache* y que ocurre con independencia del *workload* ejecutado por el cliente. En el caso de la base de datos, se observa que el tamaño de los *working sets* con los que trabaja (datos e instrucciones) son en conjunto suficientemente grandes como para que el incremento de la capacidad no tenga un efecto demasiado significativo en la tasa de fallos (cuyo valor se sitúa inicialmente en torno al 90% en la

mayoría de los *workloads*). Por el contrario, el incremento de la capacidad mitiga satisfactoriamente los fallos del cliente, reduciendo la tasa de fallos del 80% a menos del 5%, lo que indica que su *working set* tiene un tamaño más reducido. Sin embargo, al simular ambas partes de manera conjunta el resultado obtenido es una disminución notable de la tasa de fallos (enmascarando el comportamiento de la base de datos), lo que no es cierto en modo alguno para la base de datos.

La Figura 29 muestra los resultados obtenidos repitiendo el experimento con la base de datos MongoDB y revela cómo el cliente, de nuevo, distorsiona enormemente las medidas, puesto que la simulación con un único nodo arroja unos resultados en los que la tasa promedio se reduce notablemente, lo que vuelve a no ser cierto para el servidor. En este caso particular, el incremento de la capacidad tiene aún menos repercusión sobre la tasa de fallos de la base de datos que con Cassandra, puesto que la evolución de la tasa de fallos a medida que aumenta la capacidad se asemeja a una línea recta paralela al eje X. Por el contrario, la tasa de fallos del cliente evoluciona igual que con Cassandra, de un valor alto a uno despreciable. Estos experimentos llevados a cabo en la herramienta de simulación demuestran claramente la necesidad de utilizar una metodología basada en la simulación multi-nodo para este tipo de aplicaciones. El error, potencialmente provocado por el uso de una simplificación en la metodología de simulación, no puede ser, en absoluto, considerado irrelevante, ya que puede inducir a conclusiones incorrectas.

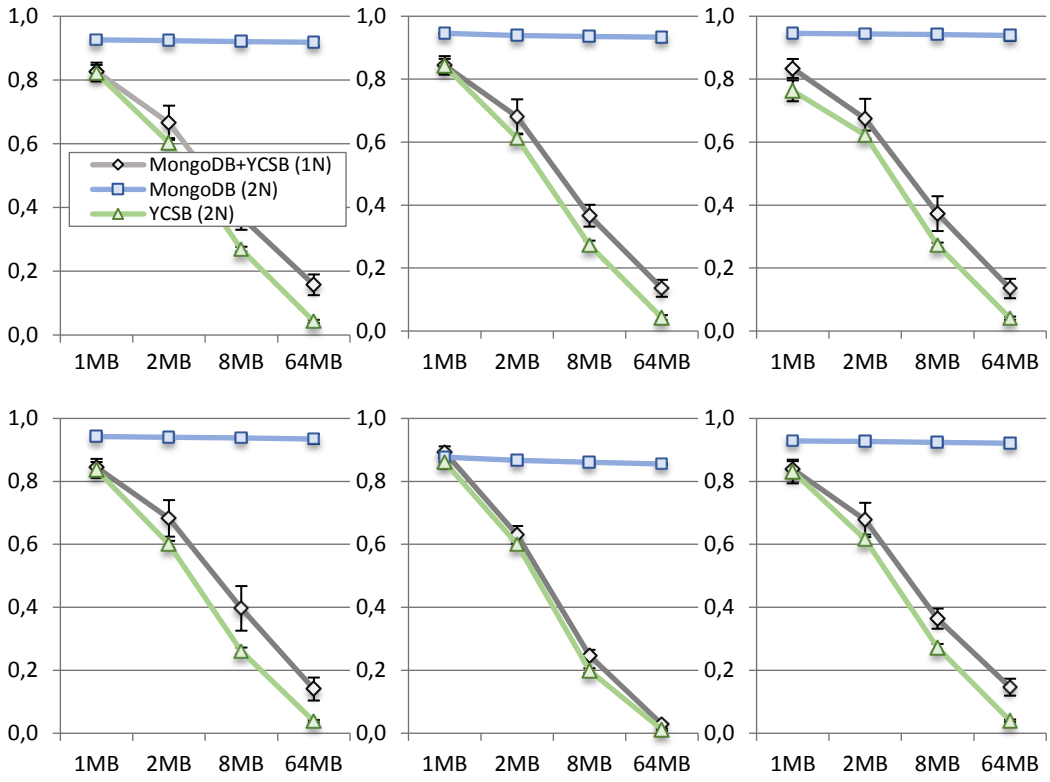

Figura 29 – Evolución del miss rate (eje Y) en LLC en función la capacidad (eje X) para la base de datos MongoDB. (arriba izquierda) Workload-A. (arriba centro) Workload-B. (arriba derecha) Workload-C. (abajo izquierda) Workload-D. (abajo centro) Workload-E. (abajo derecha) Workload-F.

Se completa esta sección cuantificando el efecto que produce la interacción de la ejecución cliente-servidor en un nodo sobre el IPC. La Figura 30 muestra el IPC que se obtiene al simular el cliente y el servidor de manera conjunta normalizado al valor del IPC del servidor cuando se simulan el cliente y el servidor por separado. Los resultados revelan una desviación positiva en un rango entre el 5% y el 10%. Este resultado se produce por la influencia del cliente al simular un

único nodo, ya que incrementa artificialmente el valor del IPC (se observa que el valor del IPC del cliente es mayor que el de la base de datos al simular ambos en un sistema).

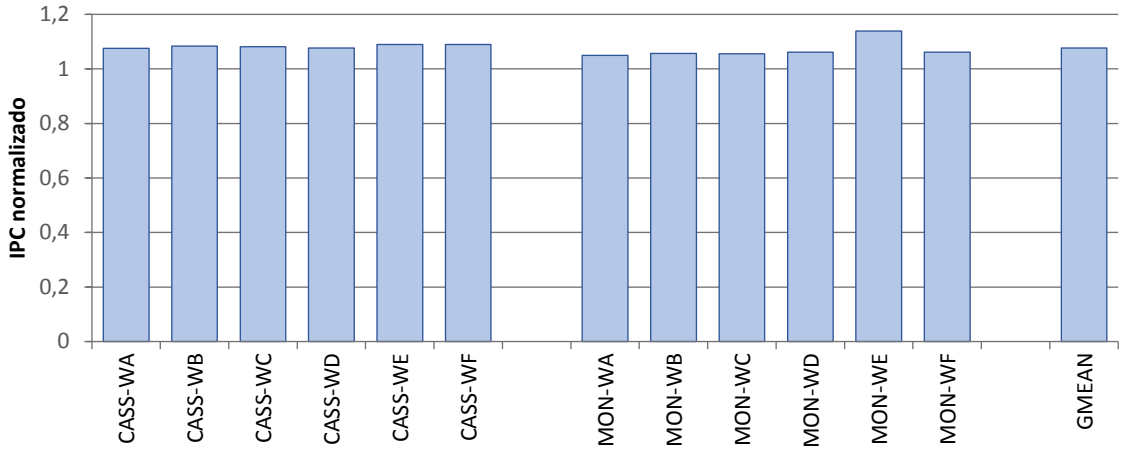

Figura 30 – IPC de las simulaciones con un solo nodo normalizado a las simulaciones multi-nodo.

Los resultados obtenidos en la monitorización de tres componentes de la micro-arquitectura revelan que el efecto de la interacción producida al ejecutar el cliente y el servidor en un solo sistema provoca un error en las medidas que no puede considerarse irrelevante, ya que puede conducir a conclusiones erróneas. La evaluación de una determinada propuesta micro-arquitectural cuyo objetivo sea la mejora del rendimiento de una base de datos como las aquí analizadas no resulta correcta si se lleva a cabo sobre un único nodo, siendo necesario adoptar metodologías como la que se ha empleado o las propuestas por [140][141].

3.3.4 *Overhead* de las Simulaciones Multi-nodo

En la última sección del capítulo se analiza el *overhead* provocado por la adaptación de la metodología para posibilitar la evaluación de las aplicaciones NoSQL, concretamente en lo concerniente a la creación de las cargas de trabajo y a su simulación. El proceso de preparación de las cargas de trabajo para este tipo de aplicaciones supone un desafío para la herramienta de simulación, puesto que requiere la creación del contenido (inicialización) para así poder realizar operaciones sobre los datos (región de interés). La creación del contenido de la base de datos se logra ejecutando un determinado número de operaciones de inserción por parte del cliente de YCSB hasta alcanzar el tamaño deseado. Este proceso de inicialización es costoso en tiempo, incluso utilizando hardware real, y resulta inabordable para el modelo *atomic* de gem5. Por este motivo, como ya se ha mencionado, se utiliza *kvm* para acelerar todo lo posible la inicialización.

Para ilustrar el esfuerzo computacional de este proceso y la necesidad de *kvm*, se lleva a cabo un experimento cuyo objetivo es la medición del tiempo necesario para cargar una base de datos utilizando tres "modelos" de CPU distintos: hardware real, *kvm* y *atomic*. Se utiliza la base de datos Cassandra con el objetivo de determinar el tiempo requerido por cada uno de los tres entornos a evaluar para la creación de dos bases de datos con un tamaño de 10MB y de 1GB. La Figura 31 recoge los resultados obtenidos, donde el eje X representa el tiempo necesario para la carga de la base de datos (minutos para el tamaño objetivo de 10MB y horas para 1GB) y el eje Y representa cada uno de los tres entornos utilizados. Los resultados revelan que tanto el rendimiento de *atomic* como el de *kvm* están lejos de la velocidad de ejecución nativa del hardware real. La generación de una base de datos con un tamaño nada realista (10MB), utilizando *atomic,* requiere un tiempo de simulación de 5 días, mientras que *kvm* consigue reducir este tiempo a varios minutos. El segundo experimento confirma que es completamente impracticable la utilización del

modelo de CPU *atomic* para cargar una base de datos con un tamaño realista (1GB), dado que el trabajo de simulación tendría una duración superior al año (los datos han sido extrapolados a partir experimento anterior). Sin embargo, *kvm* es capaz de llevar a cabo la misma tarea en menos de 24 horas, permitiendo preparar la carga de trabajo en un tiempo asumible. A raíz de los resultados de estos experimentos se puede afirmar que *kvm* es un elemento imprescindible en la metodología, no solo para las aplicaciones NoSQL sino también para las tradicionales.

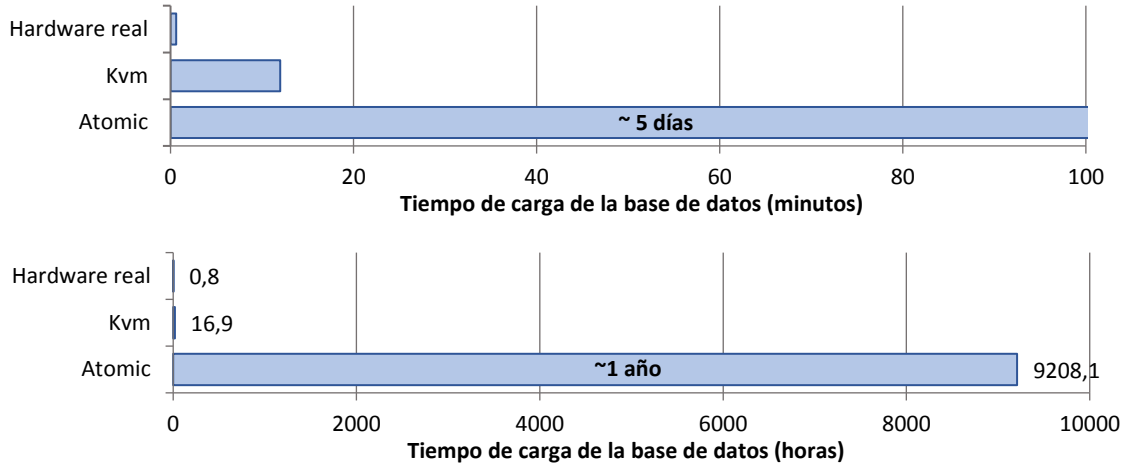

Figura 31 – Tiempo de carga de la base de datos utilizando distintas metodologías: (arriba) tamaño de 10MB (abajo) tamaño de 1GB.

A continuación, se compara el *overhead* en la metodología de preparación de las cargas de trabajo (en base a *kvm*) de las aplicaciones NoSQL frente a las aplicaciones tradicionales, midiendo el tiempo de ejecución que implica alcanzar la región de interés y crear el *checkpoint*. En el caso de las aplicaciones NoSQL, el tamaño de la base de datos que se crea es 1GB. Los resultados de las medidas están reflejados en la Figura 32. Tal y como es posible apreciar, el tiempo de preparación de las cargas de trabajo de las aplicaciones NoSQL es significativamente mayor que el de las aplicaciones tradicionales. En lo concerniente a las últimas, los *checkpoints* de las aplicaciones de NPB y PARSEC se generan, en promedio, en menos de 1 minuto, mientras que, en el caso de SPEC, esta cifra crece hasta situarse ligeramente por debajo de los 9 minutos. Cabe destacar que, debido a la heterogeneidad de las aplicaciones de SPEC, la dispersión de la medida es grande, encontrando que el tiempo de preparación de determinadas aplicaciones se sitúa por debajo del minuto, pero en determinados casos la cifra crece hasta acercarse a 1 hora. Los resultados de las aplicaciones NoSQL muestran que son bastante heterogéneos, hay una significativa variación entre las distintas bases de datos. MongoDB es la que menos tiempo tarda en completar el proceso de inicialización, situándose en torno a los 40 minutos. Para Redis y Orientdb esta cifra crece hasta situarse por encima de 5 horas y, en el caso de Cassandra, se eleva por encima de 18 horas. Los resultados de este experimento revelan que 3 de las 4 bases de datos analizadas requieren mucho más tiempo que las aplicaciones convencionales para crear las cargas de trabajo, pero que, gracias a *kvm,* es posible reducir notablemente el impacto de este proceso en la metodología, habilitando la creación de los *checkpoints* en menos de 24 horas.

Adicionalmente, se cuantifica el impacto de la simulación multi-nodo frente a la simulación de un solo nodo, en términos de tiempo de ejecución de las simulaciones y de memoria requerida. Se analiza primero el tiempo de ejecución para cada una de las bases de datos de un nodo frente a dos nodos, simulando en ambos casos el mismo número de *cores*. Así, las simulaciones con un solo nodo contienen un procesador compuesto por 8 *cores* mientras que cada nodo de las simulaciones multi-nodo contiene un procesador formado por 4 *cores*. La Figura 33 muestra el

tiempo de ejecución de las simulaciones multi-nodo de cada aplicación NoSQL normalizado al tiempo de ejecución de un solo nodo correspondiente. Los resultados revelan que no existe *overhead* en el tiempo de ejecución al simular dos nodos. Este resultado se explica por el hecho de que en ambos casos se simula la misma cantidad de instrucciones (con una ligera fluctuación), de tal manera que el coste computacional asociado es muy similar.

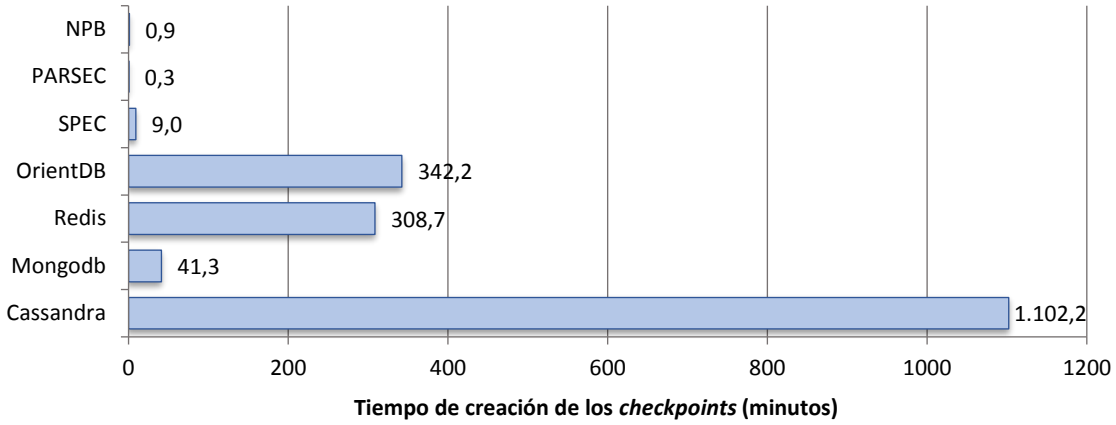

Figura 32 – Tiempo de preparación de las cargas de trabajo utilizando kvm.

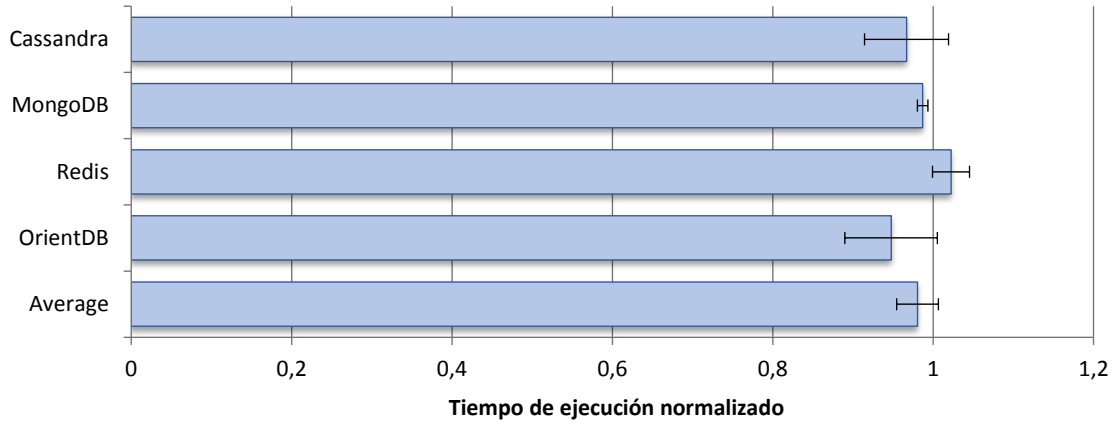

Figura 33 – Overhead de la simulación multi-nodo en términos de coste computacional (tiempo de ejecución). Resultados normalizados a los valores de la simulación de un nodo.

Por último, se cuantifica el *overhead* que implica la metodología de simulación multi-nodo en lo referente a la memoria. La cantidad de memoria que requieren los trabajos de simulación es la suma de la memoria principal que se configura para cado uno de los sistemas simulados y de la memoria que necesita el simulador para sus variables y estructuras de datos. En la metodología que se utiliza, la memoria asignada al sistema en las simulaciones con un único nodo asciende a 4GB, mientras que, en el caso de dos nodos, se configura 1GB para el nodo cliente y se mantienen 4GB para la base de datos. Además del incremento fijo de 1GB de la simulación multi-nodo frente a la simulación con un único nodo, hay que cuantificar el efecto que provoca el segundo nodo en la memoria que requiere gem5 para sus estructuras de datos internas. La Figura 34 representa la memoria total consumida por cada aplicación NoSQL normalizada a la memoria que requiere la simulación con un solo nodo correspondiente. El *overhead* en la memoria de los trabajos de simulación varía en función de la base de datos. En el caso de Cassandra y OrientDB es prácticamente despreciable mientras que para MongoDB y Redis supera el 10%. El valor medio del *overhead* se sitúa en el 7,6%, una cifra perfectamente asumible en favor de la corrección.

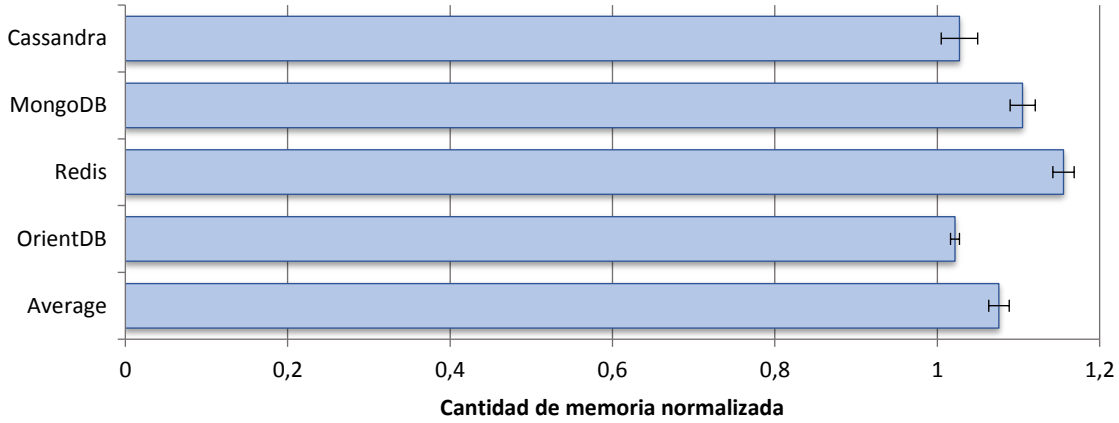

Figura 34 – Overhead de la simulación multi-nodo en términos de recursos hardware (memoria). Resultados normalizados a los valores de la simulación de un nodo.

Los experimentos han demostrado que la evaluación de aplicaciones NoSQL imponen una serie de requerimientos sobre la herramienta de simulación indispensables para garantizar la viabilidad de las simulaciones (utilización de aceleración hardware con *kvm* para generar la base de datos) y minimizar el margen de error cometido en las medidas (simulación multi-nodo para aislar la ejecución del cliente y el servidor). La preparación de gem5 para dar respuesta a los condicionantes mencionados ha supuesto un esfuerzo relevante, pero al mismo tiempo ha posibilitado la realización de trabajos como los que se describen en el resto de los capítulos de la tesis.

4 Aplicaciones NoSQL sobre Hardware de Propósito General

4.1 Introducción

El trabajo desarrollado con la metodología y la herramienta de simulación permite abordar, en este capítulo, la caracterización del comportamiento de la jerarquía de memoria empleando un conjunto de aplicaciones representativo del amplio espectro de las bases de datos NoSQL. El objetivo de este estudio es la determinación de la eficiencia del hardware de propósito general al ejecutar aplicaciones emergentes. Los trabajos previos dedicados a la caracterización de software relacionado con el *Big Data* [27][28][30][31][32] emplean en la mayoría de los casos una metodología basada en contadores hardware de rendimiento (PMU). Los experimentos realizados en estos trabajos han permitido detectar algunos cuellos de botella relevantes en la ejecución de este tipo de aplicaciones sobre procesadores de propósito general. Sin embargo, la metodología basada en contadores hardware presenta una importante limitación, consistente en la imposibilidad de realizar modificaciones a la arquitectura empleada. Esta limitación impide profundizar sobre aspectos como las capacidades de almacenamiento en chip adecuadas (jerarquía de *cache*) o el funcionamiento de mecanismos como el *prefetch hardware* o los algoritmos de reemplazo. Para poder analizar estos aspectos concretos de la jerarquía de memoria es necesario utilizar una metodología basada en simulación, como la descrita en el capítulo anterior.

En el proceso de caracterización propuesto se emplearán cada una de las bases de datos NoSQL descritas en las secciones 2.4.1 a 2.4.4, una base de datos representante por cada categoría descrita. Se utiliza el *framework* YCSB para crear el contenido de las bases de datos y para ejecutar los distintos *workloads* que incluye. Adicionalmente, se incluyen tres *suites* de aplicaciones tradicionales (SPEC, PARSEC y NPB) con el fin de comparar los resultados entre los distintos *benchmarks*. La herramienta de simulación utilizada para ejecutar estas aplicaciones es gem5 y se utiliza la configuración multi-nodo descrita en el capítulo anterior. La caracterización de todas las aplicaciones (NoSQL y "convencionales") en memoria se lleva a cabo a través de un conjunto de experimentos que comienza analizando aspectos básicos (fracción de operaciones de acceso a memoria en la mezcla de instrucciones, *working set* de instrucciones y datos) y explora, adicionalmente, aspectos más complejos como la localidad (algoritmo de reemplazo, *hardware prefetching*), la eficiencia de jerarquías multinivel o el impacto sobre los mecanismos de gestión de la compartición de datos (protocolo de coherencia). A la vista de los resultados obtenidos, a lo largo del capítulo se intentarán definir las diferencias de comportamiento, así como su posible origen, resumiendo dichas conclusiones en la última sección del presente capítulo.

4.2 Jerarquía de *Cache Single-level*

El proceso de caracterización comienza analizando los aspectos más básicos de la jerarquía de *cache*, tales como la fracción de instrucciones de acceso a memoria o los tamaños del *working set* de instrucciones y datos. Los múltiples experimentos que se van a realizar en este apartado hacen uso de una jerarquía de *cache* con un único nivel. Dicha configuración no se ajusta a la organización de la jerarquía de *cache* de un procesador comercial, siendo su único propósito la evaluación de manera adecuada del tamaño del *working set* para ambos tipos de aplicaciones.

4.2.1 Instrucciones de Acceso a Memoria

Un primer paso esencial en el estudio consiste en evaluar la relevancia de la jerarquía de memoria en el rendimiento. Si las aplicaciones presentan una fracción, de instrucciones a memoria, reducida, cualquier modificación en la jerarquía de *cache* repercutirá en menor medida sobre el rendimiento. Para ello se lleva a cabo un experimento con el objetivo de determinar el porcentaje de operaciones a memoria en la mezcla de instrucciones, mostrando la Figura 35 el número operaciones de lectura y escritura por cada 1.000 instrucciones ejecutadas (LPKI y SPKI respectivamente). En general, tanto las bases de datos como las aplicaciones convencionales presentan fracciones de referencias a memoria similares, situadas en el rango entre el 20% y el 30%. Como diferencia apreciable entre *benchmarks* convencionales y aplicaciones NoSQL, cabe destacar el mayor porcentaje de operaciones de escritura en la jerarquía de memoria de las bases de datos (SPKI), doblando prácticamente al de las aplicaciones convencionales. Dicho resultado sugiere que la eficiencia en la gestión de las operaciones de escritura puede tener un mayor impacto en este tipo de aplicaciones. Adicionalmente, resulta sorprendente la homogeneidad entre los distintos *workloads* de cada base de datos, a pesar de los distintos patrones de acceso de cada *workload* (ver Tabla 5) (WA realiza únicamente operaciones de lectura, mientras que WC mezcla lecturas y escrituras al 50%). Este resultado parece indicar que el tipo de operación sobre la base de datos no tiene un reflejo claro sobre el tipo de instrucción en el flujo de ejecución. Únicamente el *workload* E, que hace uso de un tipo especial de operaciones denominadas *scan* (las cuales acceden a varios registros secuencialmente), muestra diferencias visibles en la fracción de operaciones a memoria. Finalmente, también es destacable la similitud entre diferentes bases de datos, manteniendo todas ellas un valor de SPKI extremadamente similar a pesar de tratarse de herramientas software completamente distintas, lo que contrasta fuertemente con la variabilidad de los *benchmarks* tradicionales, donde la diferencia entre el valor mínimo y el máximo se dobla en el caso de PARSEC o, incluso se multiplica por 5 en el caso de SPEC.

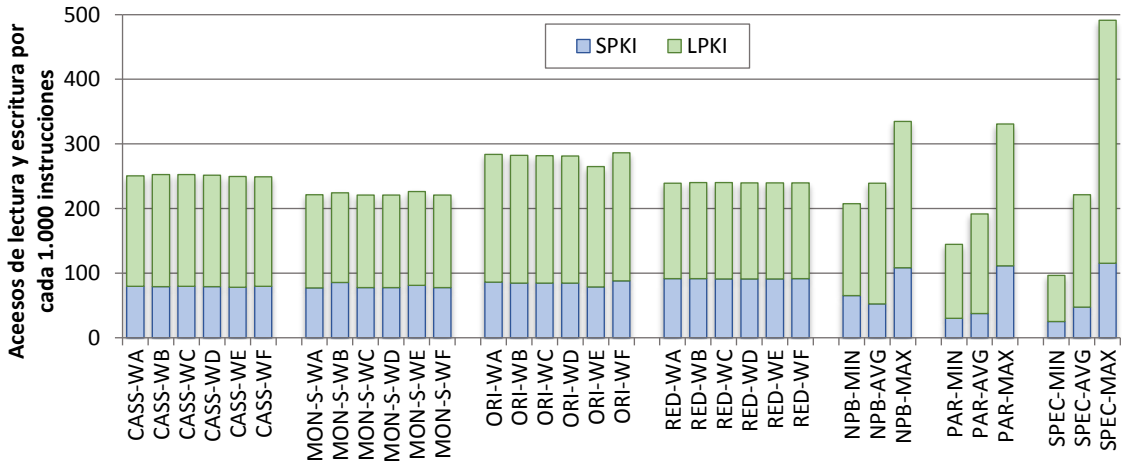

Figura 35 – Número de operaciones a memoria por cada 1.000 instrucciones (micro-ops). Únicamente se facilitan los valores máximo, mínimo y medio para los benchmarks tradicionales.

4.2.2 *Working Set* de Instrucciones

El siguiente paso de la caracterización se centra en la *cache* de instrucciones, con el objetivo de analizar la sensibilidad de las aplicaciones NoSQL al tamaño de esta estructura. Para ello, se modela en el simulador una jerarquía de *cache* con un único nivel (datos e instrucciones separados) y se incrementa la capacidad de la estructura progresivamente, desde un valor inicial de 4KB hasta alcanzar 8MB. Los incrementos de capacidad se llevan a cabo aumentando la

asociatividad con el tamaño (16KB→mapeo directo, 32KB→2 vías, 64KB→4 vías, etc.), minimizando de esta forma los conflictos causados por datos mapeados sobre la misma línea de *cache* (mismo valor de *index* pero distinto *tag*). Por último, se fija el tamaño de bloque en 64B y se utiliza una política LRU para el algoritmo de reemplazo. La Figura 36 y la Figura 37 muestran los resultados obtenidos para las aplicaciones NoSQL y para las aplicaciones convencionales respectivamente. En cada gráfica, el eje X representa los diferentes tamaños de *cache* simulados, utilizando una escala logarítmica, y el eje Y representa los fallos en la *cache* de instrucciones por cada 1.000 instrucciones ejecutadas (retiradas) o MPKI (*Miss per kilo-instruction*).

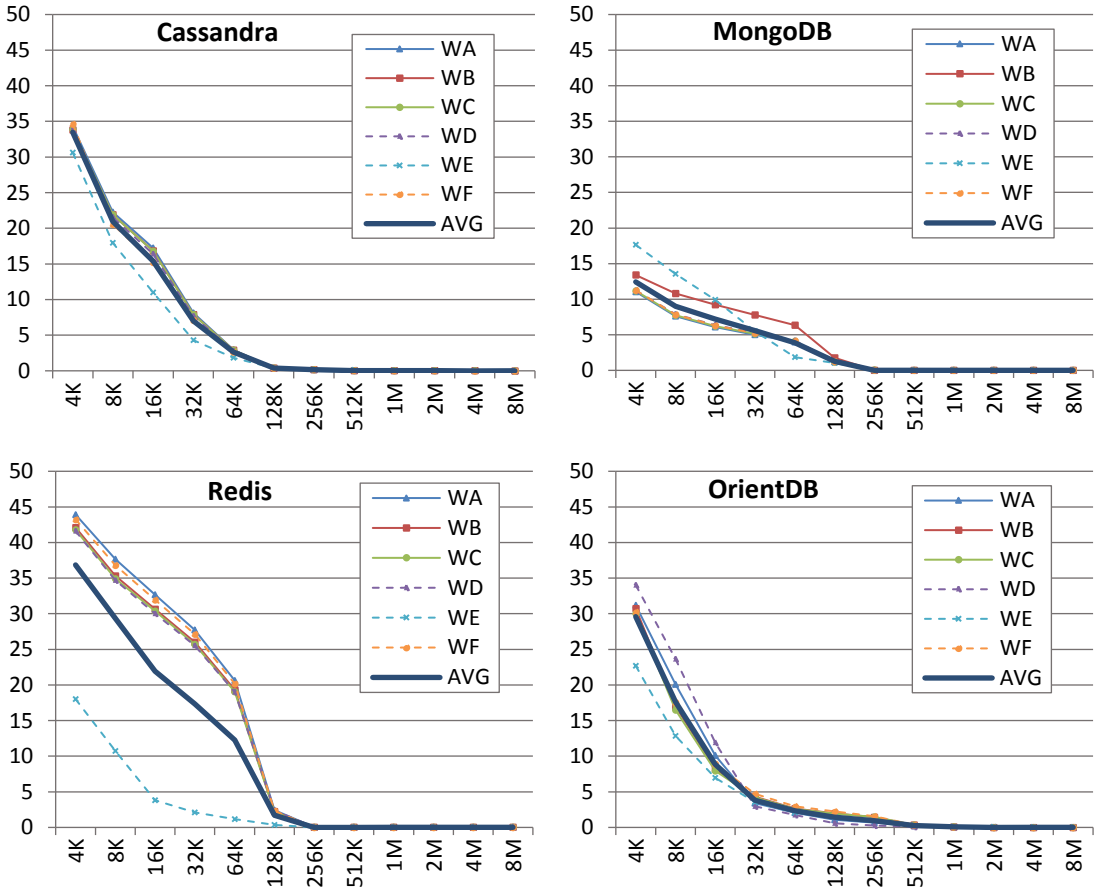

Figura 36 – Working set de instrucciones para las bases de datos NoSQL. El eje Y representa el MPKI.

Como cabe esperar, los resultados revelan que las aplicaciones NoSQL se benefician del aumento de la capacidad de la *cache* debido a la presencia de localidad en el acceso a los bloques de instrucciones, de modo que el MPKI evoluciona progresivamente a valores más pequeños. Al analizar el comportamiento de cada una de las bases de datos, es posible observar diferencias significativas. Redis muestra los valores de MPKI más elevados para un tamaño mínimo de *cache*, debido a la presencia de un porcentaje más elevado de operaciones de salto en el conjunto de instrucciones ejecutadas, reduciendo la efectividad de la *cache* de primer nivel. Con respecto al tamaño del *working set* (tamaño de *cache* donde la tendencia de MPKI se aplana), todas las aplicaciones analizadas se sitúan consistentemente en el rango de 128-256KB. Comparando ahora estos valores con los resultados obtenidos por las aplicaciones convencionales, es posible observar que las aplicaciones NoSQL alcanzan valores de MPKI más elevados. Gran parte de los *workloads* de SPEC, PARSEC y NPB presentan una tasa de fallos cercana a 0 incluso con la configuración de *cache* mínima. El tamaño del *working set* de las aplicaciones convencionales (SPEC Int: 8KB, SPEC FP: 8KB, PARSEC: 16KB, NPB: 4KB) se sitúa en un rango de 4-16KB, de modo

que las aplicaciones NoSQL presentan un *working set* de instrucciones significativamente superior al de las aplicaciones convencionales.

Este resultado confirma, de manera inequívoca, las estimaciones realizadas en trabajos previos sobre el *working set* de este tipo de aplicaciones [27][28], donde se identifica la *cache* de instrucciones como un cuello de botella significativo en las jerarquías de *cache* actuales. Sin embargo, con una metodología basada en *profiling* a través de PMU, dichos trabajos no han sido capaces de cuantificar tanto el tamaño real como la diferencia de *working set* observada entre *benchmarks* tradicionales y aplicaciones NoSQL. Gracias a la utilización de herramientas de simulación, los experimentos realizados en esta sección son capaces de eliminar dichas carencias.

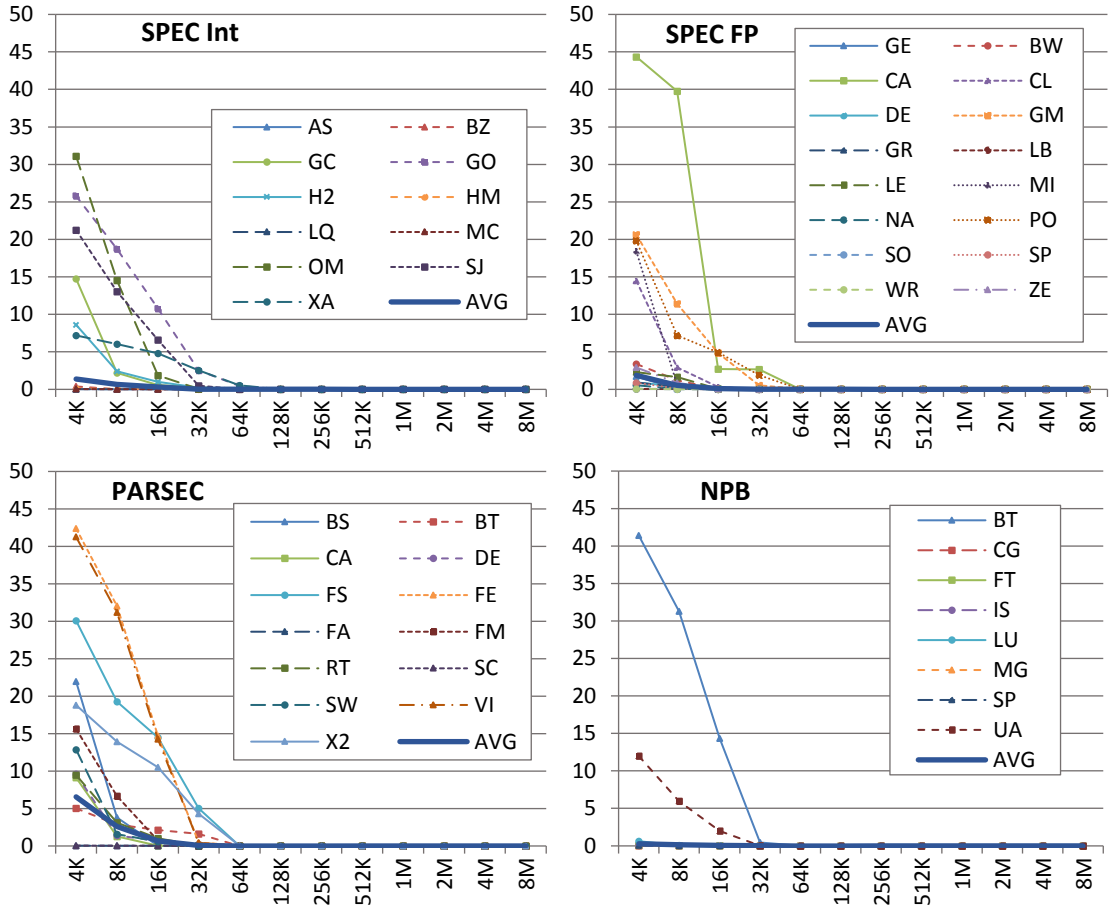

Figura 37 – Working set de instrucciones para las aplicaciones convencionales. El eje Y representa el MPKI.

4.2.3 *Working Set* de Datos

Se extiende el análisis llevado a cabo en la sección anterior a la *cache* de datos, utilizando la misma jerarquía de *cache* con un único nivel. Se realiza un experimento similar, configurando la *cache* de datos con distintos tamaños (en un rango de 16KB a 8MB) e incrementando la asociatividad con el tamaño. De nuevo, se fija el tamaño de bloque en 64B y se utiliza el algoritmo de reemplazo LRU. La Figura 38 y la Figura 39 muestran los resultados obtenidos para las aplicaciones NoSQL y para las aplicaciones convencionales respectivamente. En cada gráfica, el eje X representa los diferentes tamaños de *cache* simulados, utilizando una escala logarítmica, y el eje Y representa los fallos en la *cache* de datos por cada 1.000 instrucciones ejecutadas (retiradas) o MPKI. Cabe señalar que, en este caso, con el fin de hacer visibles los datos de todas las aplicaciones, ha sido necesario utilizar escalas distintas en el eje Y de las figuras 38 y 39.

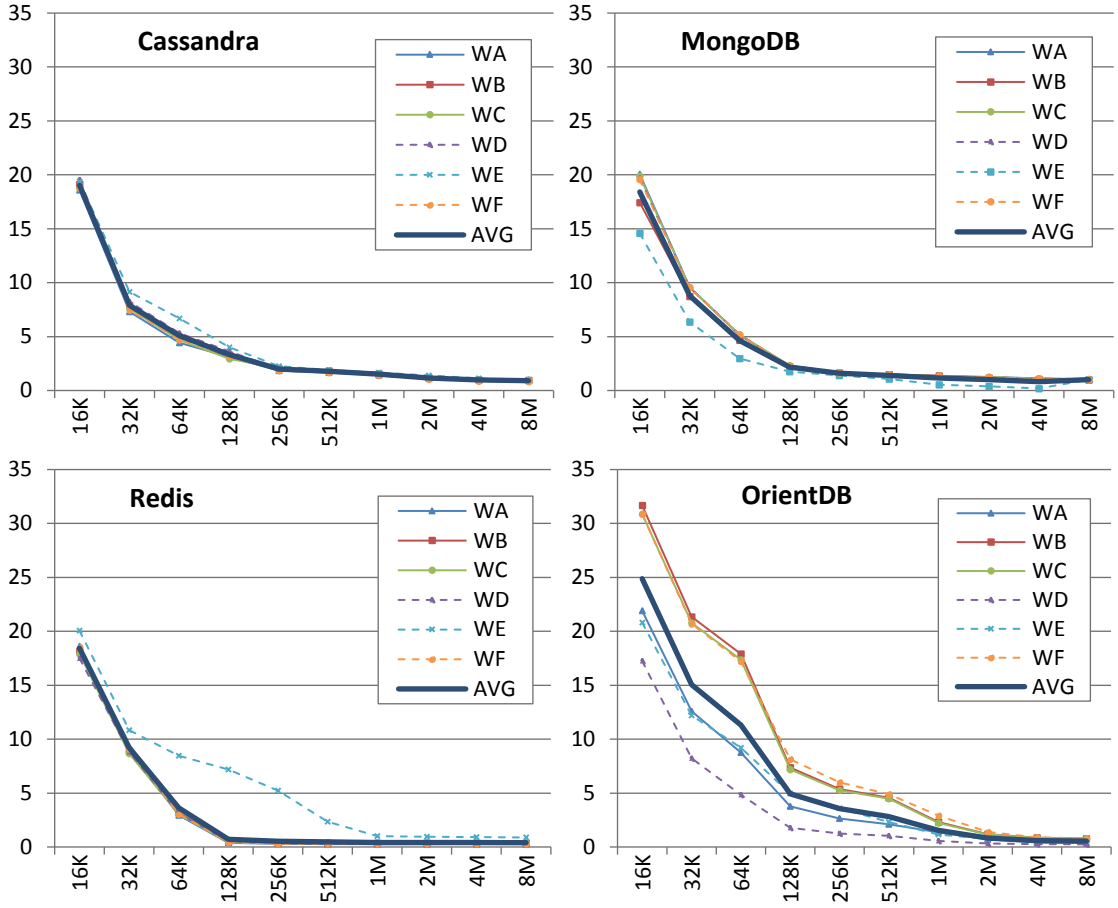

Figura 38 – Working set de datos para las bases de datos NoSQL. El eje Y representa el MPKI.

Igual que en el caso de la *cache* de instrucciones, los resultados revelan la existencia de localidad en el acceso a los datos, pues la tasa de fallos de ambos tipos de aplicaciones decrece a medida que aumenta la capacidad de la *cache*. En el caso de las bases de datos Cassandra, MongoDB y Redis, la evolución de la tasa de fallos es homogénea con independencia del *workload* simulado (salvo el *workload* E de Redis), mientras los resultados de OrientDB muestran mayores diferencias entre los distintos *workloads*, así como una tasa de fallos superior (*workloads* B, C y F). En cuanto a su valor absoluto, resulta sorprendente que instrucciones y datos presenten un *working set* similar en las aplicaciones NoSQL, contrastando fuertemente con las diferencias de tamaño habituales en aplicaciones convencionales. En cualquier caso, el valor promedio de la tasa de fallos es sorprendentemente similar para el conjunto de bases de datos y decrece exponencialmente con independencia de la base de datos o del *workload* simulado. Por el contrario, al analizar los resultados de las aplicaciones convencionales se observa un comportamiento más heterogéneo (propio de aplicaciones tan dispares como las que se agrupan en cada *suite*), mostrando *working sets* que exceden claramente el límite superior establecido (*streamcluster*, *libquantum* o *LU*), junto con otros de un tamaño muy reducido. Sin embargo, cabe destacar que la evolución de los valores medios (media geométrica) de ambos tipos de aplicaciones es similar, identificando en la gran mayoría de los casos tamaños de *working set* en el rango de los 4-8MB.

Los resultados de los experimentos contradicen las altas tasas de fallos obtenidas para la *cache* de datos en [30]. Tras varios intentos por replicar dichos resultados y un análisis de los mismos, parece clara la existencia de algún tipo de error en ese trabajo previo. Dichos resultados muestran valores de MPKI superiores a 150, lo que combinado con unos valores de SPKI+LPKI cercanos a 250 (ver Figura 35) implicaría un *miss rate* de L1 superior al 50%, un valor altamente improbable.

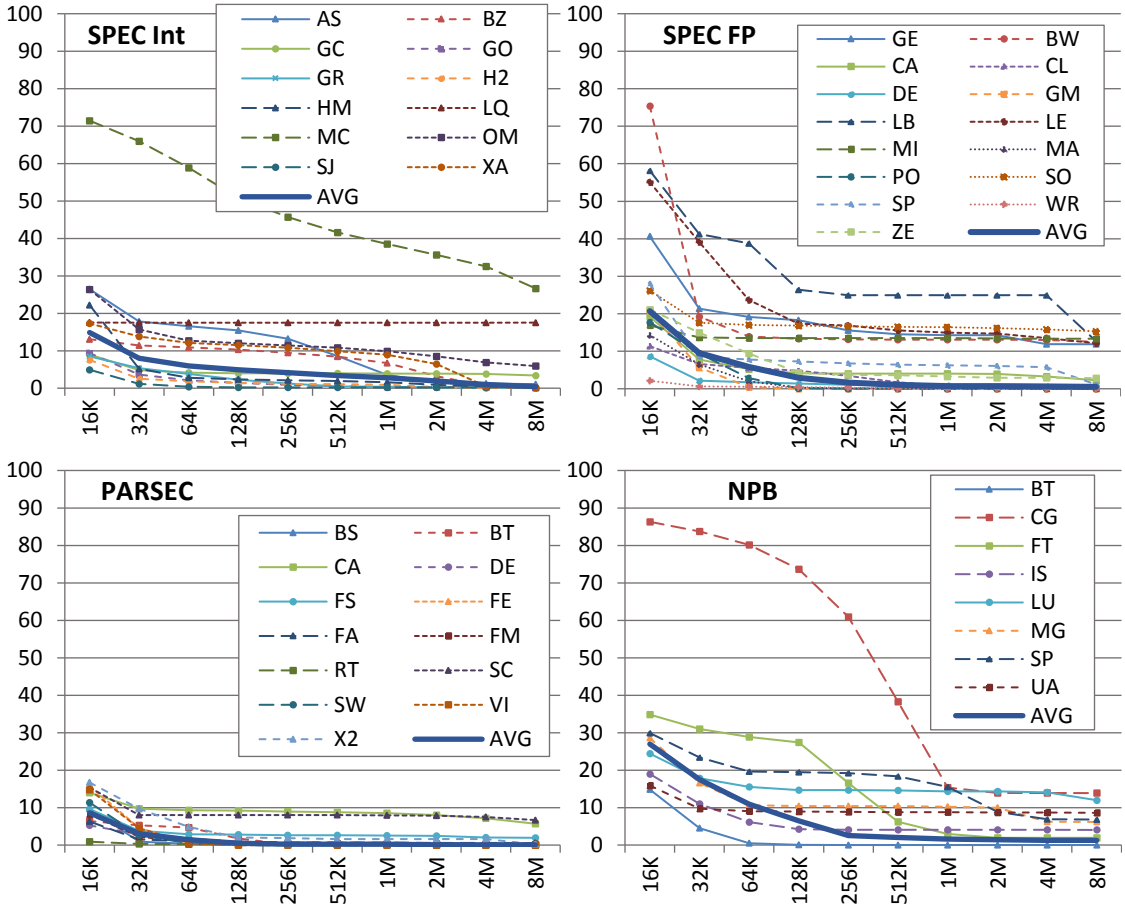

Figura 39 – Working set de datos para las aplicaciones convencionales. El eje Y representa el MPKI.

Los experimentos anteriores han revelado un resultado un tanto sorprendente, puesto que el tamaño del *working set* de datos de las aplicaciones NoSQL y de las convencionales es similar, a pesar de la diferencia en el tamaño del problema de cada tipo de aplicación. Así, mientras que el tamaño de las bases de datos con las que se ha trabajado excede el GByte, el tamaño medio de los ficheros de entrada de las aplicaciones SPEC y PARSEC se sitúa en un rango comprendido entre el 0,5MB y 20MB. Estos resultados sugieren que el tamaño de la base de datos no tiene relación con el *working set* de datos. Para confirmar este hecho se lleva a cabo un experimento adicional, analizando la evolución del MPKI con distintos tamaños de la base de datos (32MB, 512MB, 1GB y 2GB). Se varía el tamaño de la *cache* desde 16KB hasta 8MB al igual que en el experimento anterior y se evalúa, cada configuración de la *cache* de datos, para cada base de datos, *workload* y tamaño de base de datos. La Figura 40 muestra parte de los resultados (Cassandra y MongoDB, *workloads* A y C), observando una tendencia similar en el resto de *workloads* y bases de datos. Es posible apreciar cómo la tasa de fallos evoluciona de manera simular con independencia del tamaño de base de datos, aun cuando el tamaño más grande utilizado multiplica por 64 el tamaño más pequeño.

La drástica disminución de los valores de MPKI para el tamaño de *cache* más grande sugiere que el tamaño de *working set* puede tener más relación con el tamaño de los *records* que con el de la base de datos. Por este motivo, se extiende el análisis para determinar el efecto del tamaño de los *records* en el *working set* de datos. La configuración por defecto del tamaño de los *records* especifica que cada uno está compuesto por un total de 10 campos, donde cada campo tiene un tamaño de 100B, de modo que el tamaño total asciende, aproximadamente, a 1KB. Por otra parte, la extensión de las operaciones de lectura y escritura es distinta. Las lecturas acceden a la totalidad

de los datos, mientras que la escrituras solamente modifican un campo del *record*. Se evalúan dos tamaños adicionales para los campos: 10B y 100KB, de modo que el tamaño total de los *records* asciende a 100B y 1MB respectivamente. Se varía el tamaño de la *cache* desde 16KB hasta 8MB, al igual que en los experimentos anteriores, y se evalúa cada configuración de la *cache* de datos para cada base de datos, *workload* y tamaño de base de datos.

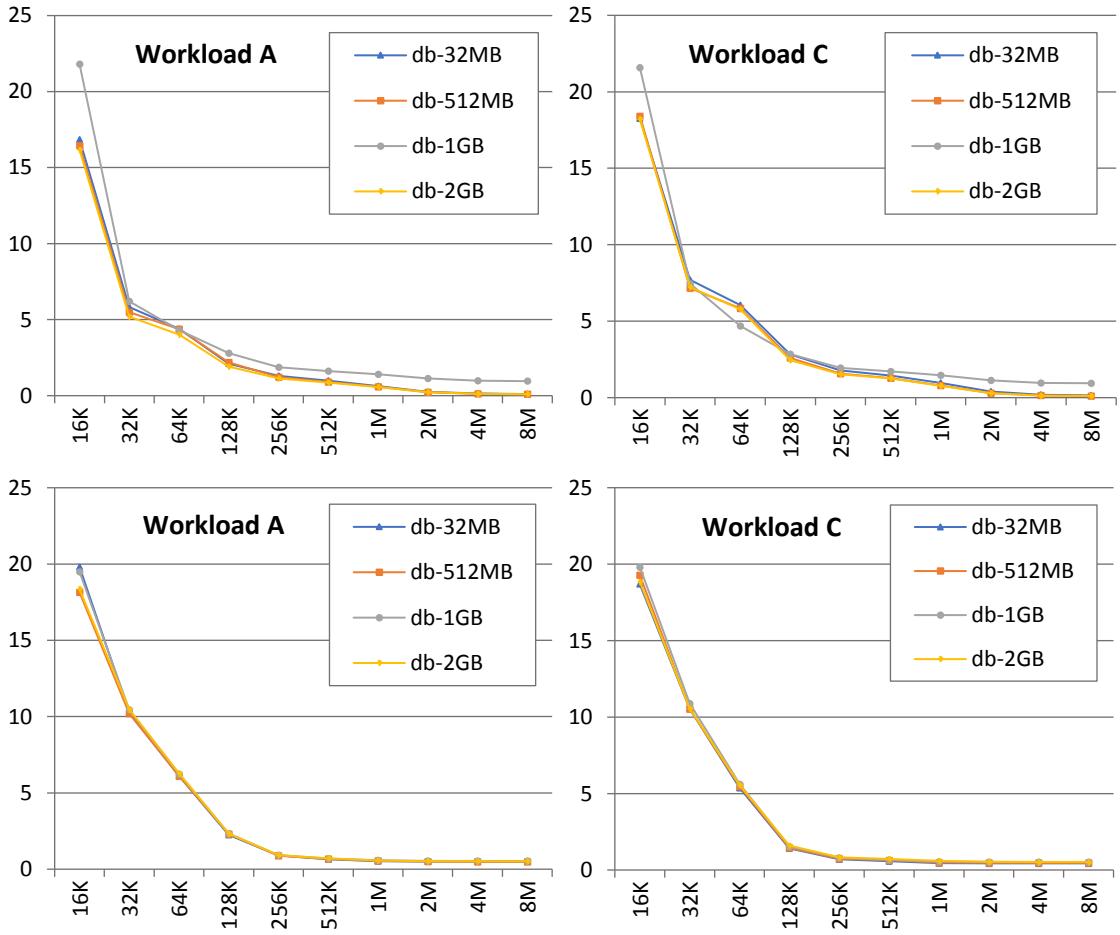

Figura 40 – Evolución del working set de datos para distintos tamaños de bases de datos. El eje Y representa el MPKI. (arriba) Cassandra. (abajo) MongoDB.

Los resultados de la Figura 41 muestran que, en el caso de la base de datos Cassandra, el efecto de la variación del tamaño de los *records* es despreciable, puesto que la diferencias entre los tres tamaños evaluados son mínimas. Sin embargo, este resultado no es extensible a MongoDB, puesto que las diferencias se acentúan con el incremento del tamaño del registro. Por una parte, los valores del MPKI para el tamaño más grande evaluado son significativamente mayores y únicamente comienzan a disminuir cuando la capacidad de la *cache* es mayor que el propio *record*. Por otra parte, también existe cierta diferencia entre los restantes tamaños evaluados para las configuraciones más pequeñas de *cache*, llegando un punto en el que tales diferencias desaparecen. Este experimento ha revelado que el tamaño de los *records* tiene cierto efecto en el MPKI en determinadas bases de datos.

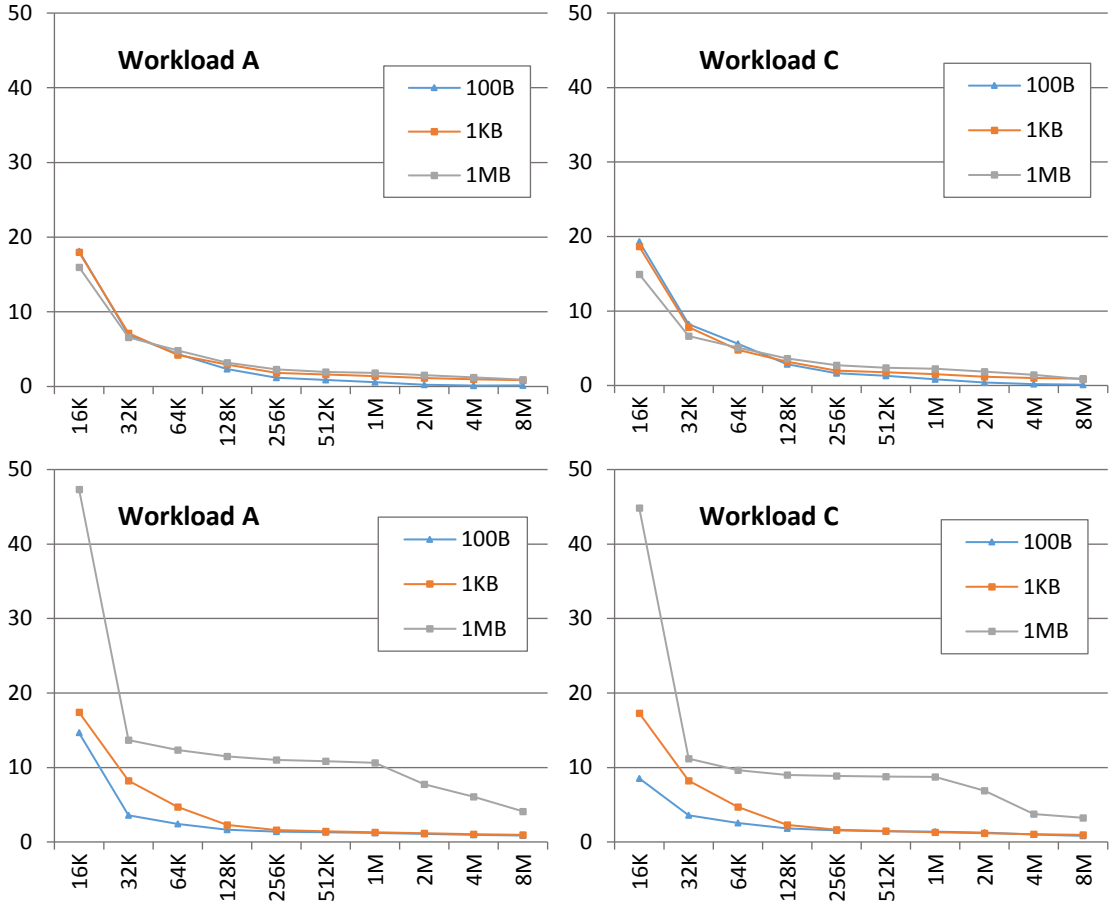

Figura 41 – Evolución del working set de datos para distinto tamaño de los records. El eje Y representa el MPKI. (arriba) Cassandra. (abajo) MongoDB.

Con el objetivo de entender mejor la relación entre la base de datos y el *working set*, se dedica un último experimento a analizar el efecto del patrón de acceso, a los *records* de la base de datos, sobre la efectividad de la jerarquía de *cache* (localidad espacial). Por defecto, los *workloads* de YCSB utilizan dos distribuciones que presentan un alto nivel de localidad en los datos accedidos, la distribución *latest* para el *worload* D y la distribución *zipfian* para el resto de *workloads*. Se lleva a cabo un experimento, similar a los anteriores, para medir la evolución del MPKI cuando se modifica el patrón de acceso de los *workloads* por uno carente de localidad, consistente en un acceso aleatorio con una distribución uniforme. Los resultados de MongoDB, mostrados en la Figura 42, revelan que las diferencias entre los distintos patrones de acceso son mínimas. El único resultado reseñable tiene lugar con el *workload* E, donde la distribución uniforme eleva ligeramente el número de fallos.

Se amplía este experimento extendiendo la evaluación a todos los patrones de acceso incluidos en YCSB, un total de seis distribuciones: *zipfian*, *latest*, *uniform*, *hotspot*, *sequential* y *exponential*. Se limitan los resultados a los obtenidos para el *workload* A, mostrados en la Figura 43. Se observa claramente que la influencia de la distribución de acceso a los datos es despreciable, únicamente apreciando mínimas diferencias en las configuraciones de *cache* con tamaños más pequeños. Se puede concluir, en consecuencia, que tanto el tamaño de la base de datos como el patrón de acceso a los mismos tienen una influencia residual sobre la tasa de fallos en la *cache* de datos. Resulta complicado inferir una explicación precisa a estos resultados. El complejo *stack* software que utilizan las aplicaciones NoSQL parece alejar, de manera drástica, las operaciones de lectura/escritura (*read*/*modify*) sobre la base de datos de las instrucciones de lectura/escritura

(*load*/*store*) del procesador, amortiguando u ocultando los efectos de las distintas pruebas realizadas sobre la jerarquía de memoria. Únicamente se han constatado diferencias reseñables en el MPKI al modificar el tamaño de los *records*, mientras que en el resto de los experimentos los resultados han sido sorprendentemente homogéneos.

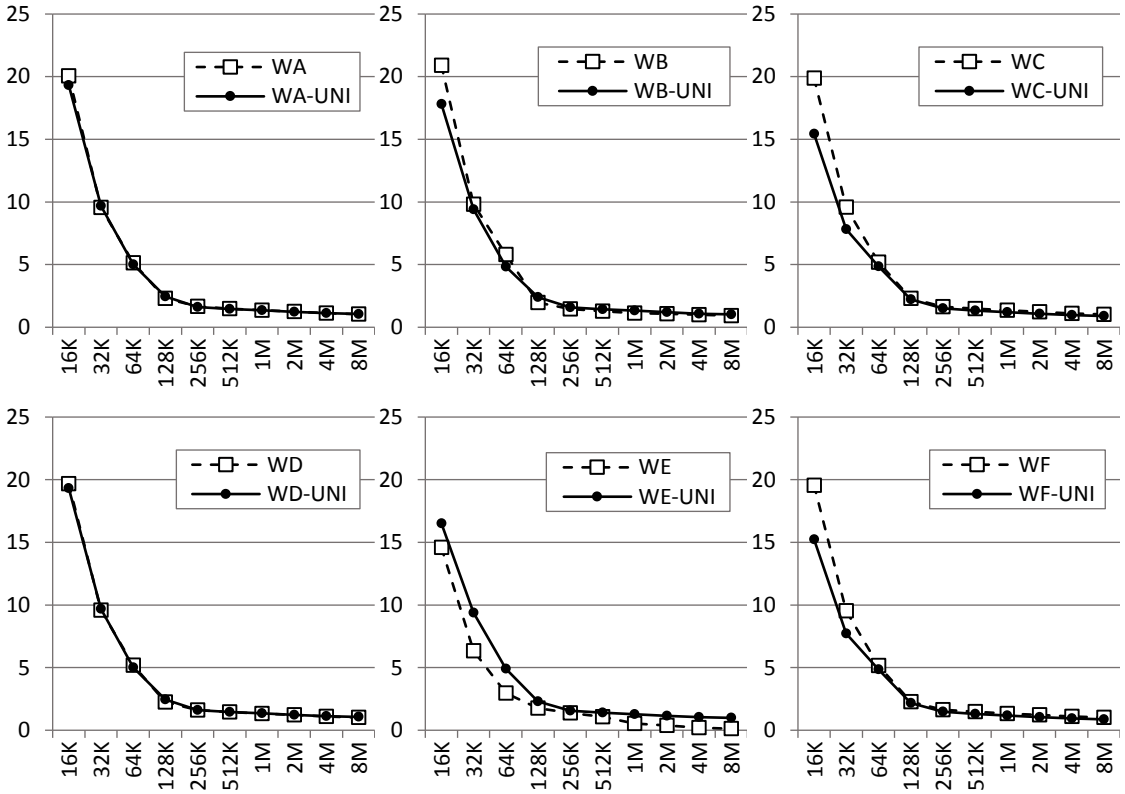

Figura 42 – Evolución del working set de datos comparando las distribuciones de acceso a los datos configuradas por defecto y la distribución uniforme. El eje Y representa el MPKI. Resultados para MongoDB.

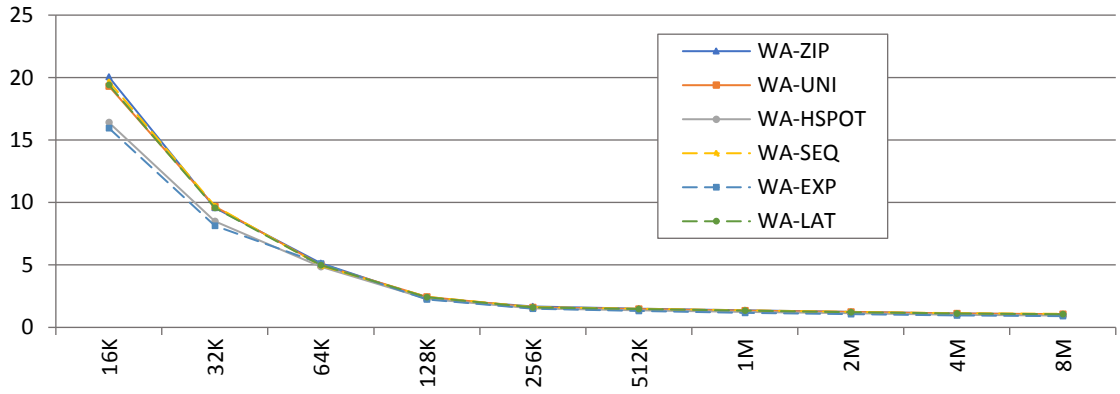

Figura 43 – Evolución del working set de datos comparando distintas distribuciones de acceso a los datos. El eje Y representa el MPKI. Resultados para MongoDB.

4.3 Jerarquía de *Cache* Multinivel

Una vez analizados los aspectos más básicos respecto a la interacción de las aplicaciones NoSQL con la jerarquía de *cache*, se abandona la jerarquía *single-level* en favor de una configuración más realista compuesta por tres niveles (cuyos principales parámetros se resumen en la Tabla 6), que trata de asemejarse a la existente en un procesador comercial actual. La caracterización del

comportamiento de las aplicaciones en esta configuración de *cache* multinivel comienza evaluando el MPKI en los distintos niveles, así como la sensibilidad a variaciones de tamaño en cada nivel. A continuación, se evalúa la eficiencia de mecanismos presentes en la mayoría de los procesadores actuales, tales como la política de reemplazo de bloques, los mecanismos de *prefetch hardware* o aspectos relativos al protocolo de coherencia.

Tabla 6 – Configuración de la jerarquía de cache multinivel.

Caches privadas	(L1) Capacidad/Asociatividad/Tamaño de bloque/Tiempo de acceso	32KB I/ 32KB D / 8 / 64B / 1 ciclo
	(L2) Capacidad/Asociatividad/Tamaño de bloque/Tiempo de acceso	256KB/8/64B/4 ciclos / Exclusiva con L1
Cache compartida	Capacidad/Asociatividad /Tamaño de bloque / Tipo	8MB, 16, 64B, *Mostly Inclusive*
	NUCA Mapping	*Static, interleaved by LSB*
	Protocolo de coherencia	*MOESI snooping*
	Data Slice Size/Tiempo de acceso	2MB / 6 ciclos
Mem	Capacidad / Tiempo de acceso / Ancho de banda	4GB /240 ciclos / 32GB/s

4.3.1 MPKI en la Jerarquía de Memoria

La evaluación de la tasa de fallos en cada uno de los niveles de la jerarquía se presenta en la Figura 44, donde el eje Y muestra los fallos por cada mil instrucciones. En todas las gráficas aparecen los resultados de MPKI de la configuración resumida en la Tabla 6 (etiquetados como "BASE"), así como configuraciones alternativas doblando el tamaño del primer nivel de *cache* (instrucciones y datos, etiquetada como "L1X2"), y doblando el tamaño de la LLC (etiquetada como "L3X2"). El resto de los parámetros de la configuración (asociatividad, tiempo de ciclo, etc.) se mantienen constantes.

Los resultados de la *cache* de datos de primer nivel muestran tasas de fallo próximas entre aplicaciones convencionales y NoSQL, confirmando las observaciones previas sobre su similar *working set* de datos. Por ser la aplicación con un *working set* más significativo, OrientDB presenta los valores de MPKI más elevados, llegando a doblar los obtenidos por el resto de las aplicaciones. En lo concerniente a la *cache* de instrucciones, se puede observar de nuevo la diferencia existente entre el *working set* de instrucciones de las aplicaciones NoSQL y las aplicaciones tradicionales. De hecho, las bases de datos MongoDB y Redis tienen incluso más fallos en esta estructura que en la *cache* de datos. De nuevo, se confirman los resultados del análisis previo sobre el tamaño del *working se*t de instrucciones. Tanto en el caso de los datos como de las instrucciones, se observa una evolución similar al duplicar su capacidad, tal y como se observó en los experimentos anteriores.

Los resultados obtenidos para el siguiente nivel de la jerarquía (L2) revelan un efecto secundario asociado al incremento de capacidad de las *caches* de primer nivel (BASE vs. L1X2). Este incremento de tamaño en L1 provoca una mejora en los valores de MPKI de las aplicaciones NoSQL, un efecto que no se traslada en la misma medida a las aplicaciones convencionales (salvo SPEC). Cuando se analizó el *working set* de datos, se observó una evolución del MPKI mucho más progresiva en el caso de las aplicaciones NoSQL, frente a los abruptos escalones mostrados por algunas aplicaciones convencionales. Este descenso progresivo parece ayudar a la L2, pues la capacidad agregada (L1+L2) es mayor y el MPKI de L2 disminuye.

Finalmente, en el último nivel de *cache* no se aprecia un efecto similar al observado en L2, desapareciendo cualquier mejora en la tasa de fallos asociada al incremento de tamaño de L1. El incremento de capacidad en este nivel parece tener un efecto más uniforme sobre todas las aplicaciones, provocando una reducción de la tasa de fallos para ambos tipos que llega, en algún

caso, hasta el 30%. Todas las aplicaciones, NoSQL y convencionales, muestran similares niveles de localidad temporal (al igual que sucede en L1), lo que les permite aprovechar la capacidad extra de LLC para así mejorar su tasa de fallos.

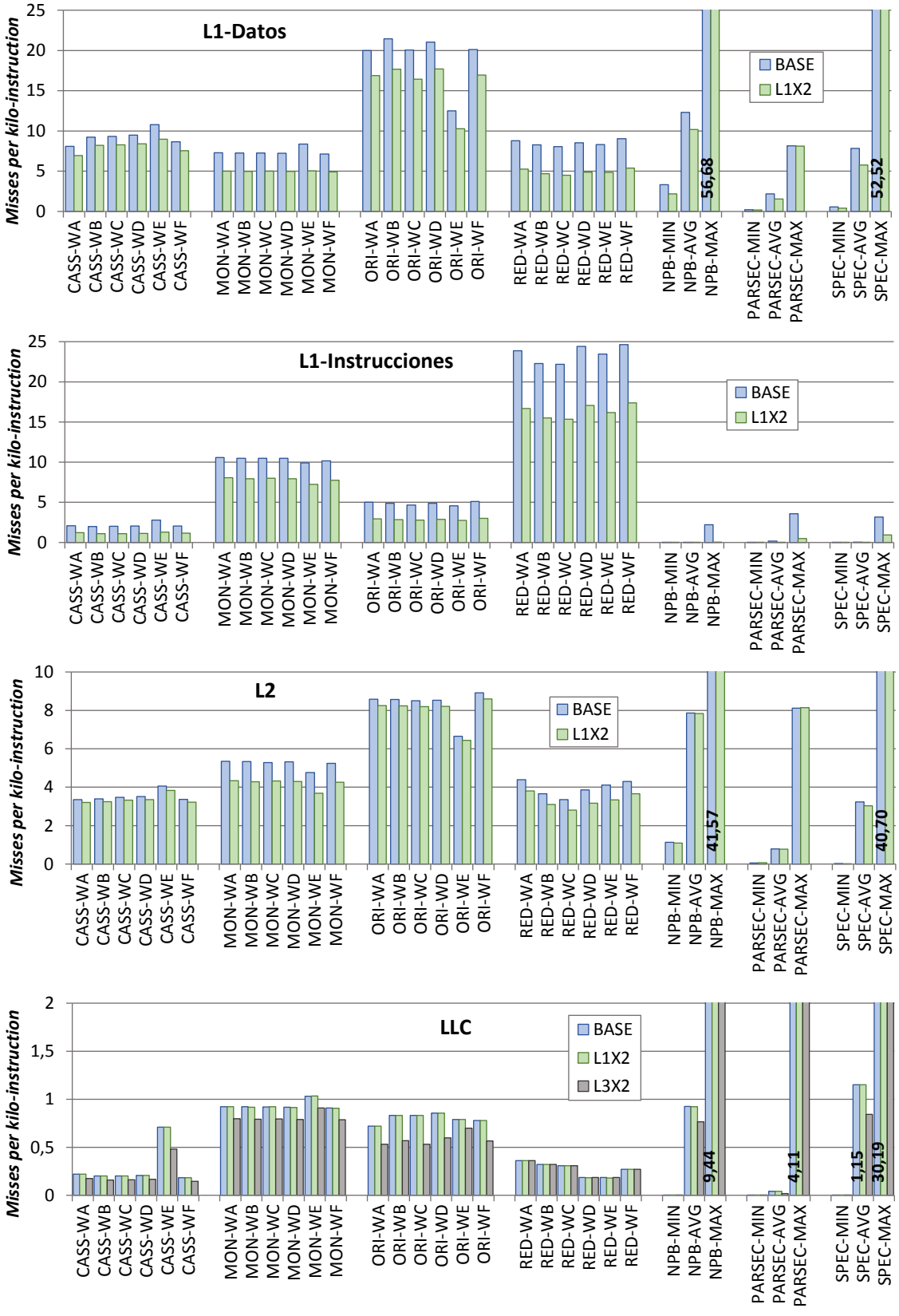

Figura 44 – Rendimiento de la jerarquía de cache de tres niveles.

4.3.2 Algoritmo de Reemplazo

Para analizar y comparar el grado de localidad temporal de las bases de datos NoSQL y de las aplicaciones convencionales se evalúa la tasa de fallos utilizando dos políticas de reemplazo distintas. Se hace uso del algoritmo LRU (*Least Recently Used*) y de otro basado en la selección aleatoria del bloque que se va a reemplazar (etiqueta "RAND"). La evaluación se lleva a cabo, tanto en la *cache* de datos de primer nivel (L1D) como en el último nivel de *cache* (LLC). En ambos experimentos, los niveles donde no se realizan medidas de tasa de fallo utilizan LRU (en la evaluación de LLC con RAND las *caches* de primer nivel y de segundo nivel utilizan LRU). La Figura 45 muestra los resultados de las evaluaciones, donde aparecen los resultados de LRU normalizados a los valores obtenidos con RAND.

Los valores observados para la *cache* de datos de primer nivel (L1D) indican que tanto las aplicaciones NoSQL como las convencionales tienen un grado de localidad temporal similares, siendo LRU consistentemente mejor que RAND. En el caso de las aplicaciones NoSQL, MongoDB es la base de datos que mejor rendimiento extrae de LRU, con una mejora del 10% aproximadamente, mientras que en el resto de las bases de datos ésta se sitúa alrededor del 5%. Los resultados son además muy homogéneos para todos los *workloads*, siendo el mejor resultado obtenido por el *workload* E (apreciable de manera significativa para Redis), gracias al tipo de operaciones de *scan*, que parecen presentar una localidad temporal ligeramente superior en los accesos a memoria. En el caso de las aplicaciones convencionales, el valor medio de la mejora de LRU sobre RAND es ligeramente superior.

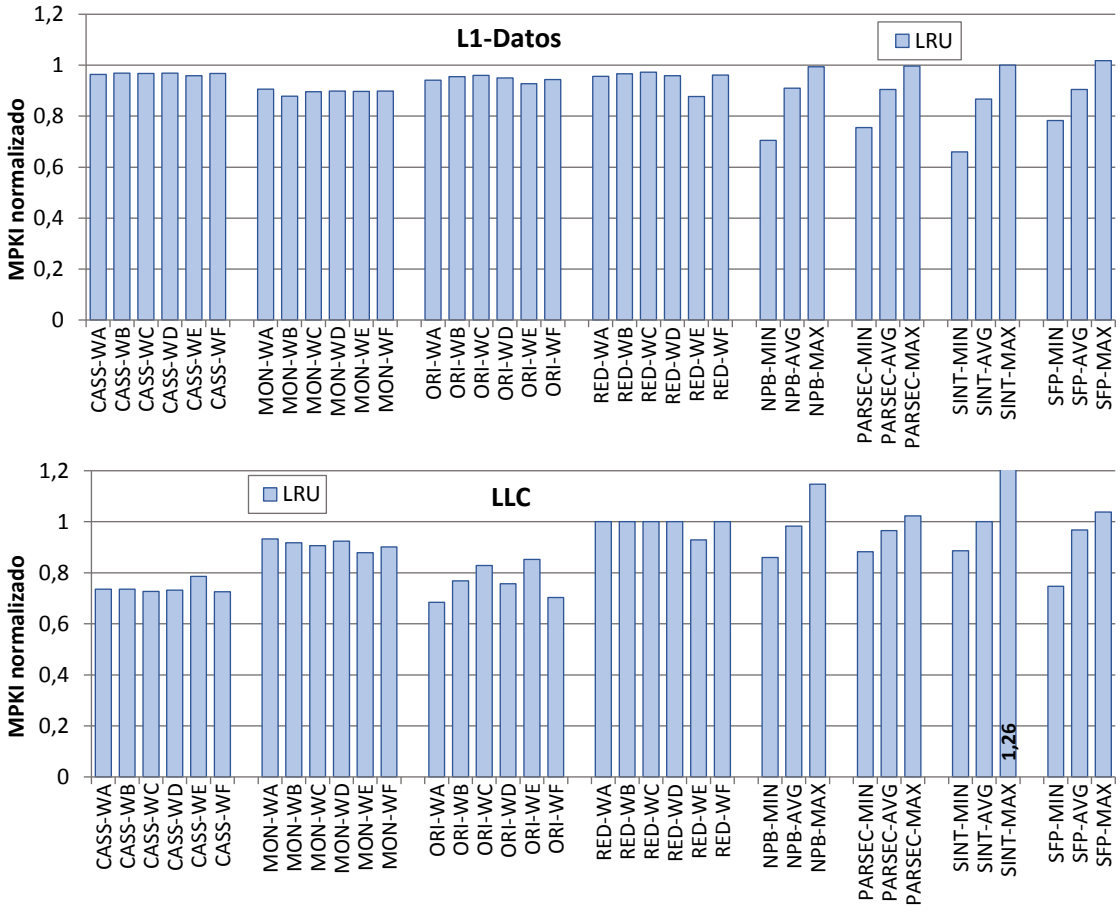

Figura 45 – Efecto de la política de reemplazo en el MPKI. Los resultados están normalizados a los valores de RAND.

Contrariamente, los resultados del experimento en el último nivel de *cache* muestran mayores diferencias entre el conjunto de aplicaciones que en los obtenidos en L1D. En el caso de las aplicaciones convencionales, la localidad temporal ha sido filtrada en su mayor parte por los niveles superiores en muchas de las aplicaciones, de tal forma que el impacto de LRU es mucho menor. Así, la mejora de LRU sobre RAND se reduce a menos del 3% en LLC y, además, crece significativamente el número de aplicaciones que funcionan peor con el algoritmo LRU que con la selección aleatoria del bloque a reemplazar. Sin embargo, en el caso de las aplicaciones NoSQL, se observa que hay suficiente localidad temporal como para que LRU consiga mejores resultados de manera consistente en comparación con RAND. De hecho, el beneficio crece notablemente con respecto a L1 en el caso de Cassandra y OrientDB, mientras que se mantiene y empeora para MongoDB y Redis respectivamente. Las diferencias encontradas al evaluar el funcionamiento del algoritmo LRU para ambos tipos de aplicaciones sugieren que las aplicaciones NoSQL tienen más localidad en el último nivel de *cache* que las aplicaciones tradicionales, de modo que LRU consigue reducir satisfactoriamente la tasa de fallos para 3 de las 4 aplicaciones evaluadas. Esta relación con el algoritmo de reemplazo sugiere una revisión del rendimiento de algoritmos alternativos propuestos en la literatura para estas aplicaciones, siendo un mecanismo más crítico para su rendimiento.

4.3.3 Hardware Prefetching

Se extiende el análisis de la localidad espacial de las aplicaciones realizando un experimento similar al anterior. Se mide en este caso el MPKI al utilizar dos algoritmos de *prefetching* distintos y se comparan los resultados obtenidos con aquellos que se obtienen en ausencia de *prefetching*. El primer algoritmo es *Tagged Prefetcher* [142], el cual genera una referencia al siguiente bloque secuencial cada vez que se produce un fallo en la *cache* o cada vez que un bloque que reside en la *cache* debido a un *prefetch* es referenciado por primera vez, de modo que, si el bloque no está presente en el nivel donde se aplica el algoritmo se pide al siguiente nivel de la jerarquía de memoria. El segundo algoritmo, denominado *Arbitrary Stride Prefetcher* [143], detecta *strides* (secuencias de referencias a memoria) de longitud constante (accesos a *arrays* generados en bucles) en las referencias que generan los *loads*/*stores* utilizando una estructura hardware que almacena parte de la historia pasada para determinar qué bloque traer antes de que sea referenciado por una instrucción de acceso a memoria. Ambos *prefetchers* son configurados con un grado de 2 bloques, es decir, generan peticiones para llevar al nivel de la jerarquía de memoria correspondiente los 2 siguientes bloques que son susceptibles de ser utilizados en un futuro. Al igual que en el experimento anterior, se evalúa la localidad espacial en la *cache* de datos de primer nivel (L1D) y en el último nivel de *cache* (LLC), de tal manera que únicamente se utiliza una política de *prefetch* hardware en uno de los dos niveles a evaluar, mientras que en el resto de la jerarquía no se hace uso de *prefetching.* La Figura 46 muestra los resultados normalizados a los valores obtenidos en ausencia de mecanismos de *prefetch*.

Los resultados en la *cache* de datos de primer nivel revelan un resultado dispar en el caso de las aplicaciones NoSQL. Tanto Cassandra como Redis obtienen malos resultados con el uso de ambos *prefetchers,* siendo los mecanismos de *prefetching* incapaces de compensar la polución que introducen. De manera opuesta, tanto MongoDB como OrientDB obtienen mejoras en el MPKI significativas, de hasta el 20% y el 60% respectivamente. En este nivel de la jerarquía de memoria, ambos *prefetchers* obtienen un rendimiento razonable en el caso de las aplicaciones convencionales. La mejora es notable en aquellas aplicaciones con mucha localidad espacial, pero en aquellas con poca localidad también es posible observar pérdidas de rendimiento, tal y como atestigua la diferencia entre los valores máximo y mínimo.

La disparidad de resultados entre las aplicaciones NoSQL y las convencionales se reduce notablemente en la LLC, donde el rendimiento en presencia de ambos *prefetchers* mejora de manera consistente para la gran mayoría de aplicaciones. Gracias a la capacidad agregada del último nivel de *cache*, la polución deja de tener un efecto tan adverso, mientras que la elevada penalización de los accesos fuera del chip aumenta el efecto positivo de cada bloque pre-cargado de manera correcta. De nuevo, es posible observar cómo el rendimiento de las políticas de *prefetching* es altamente dependiente de la aplicación, tanto en el caso de las NoSQL como de las convencionales. La uniformidad observada en los experimentos previos entre los distintos *workloads* de cada base de datos desaparece en este caso. Solamente MongoDB es capaz de mantener un comportamiento uniforme, observando en el resto de las bases de datos una notable variabilidad entre *workloads*. Resulta evidente que los mecanismos de *prefetching* son extremadamente sensibles al flujo de ejecución de la aplicación, y diferencias que pasan desapercibidas a otros mecanismos de la *cache* afectan al comportamiento del *prefetcher* de manera notable.

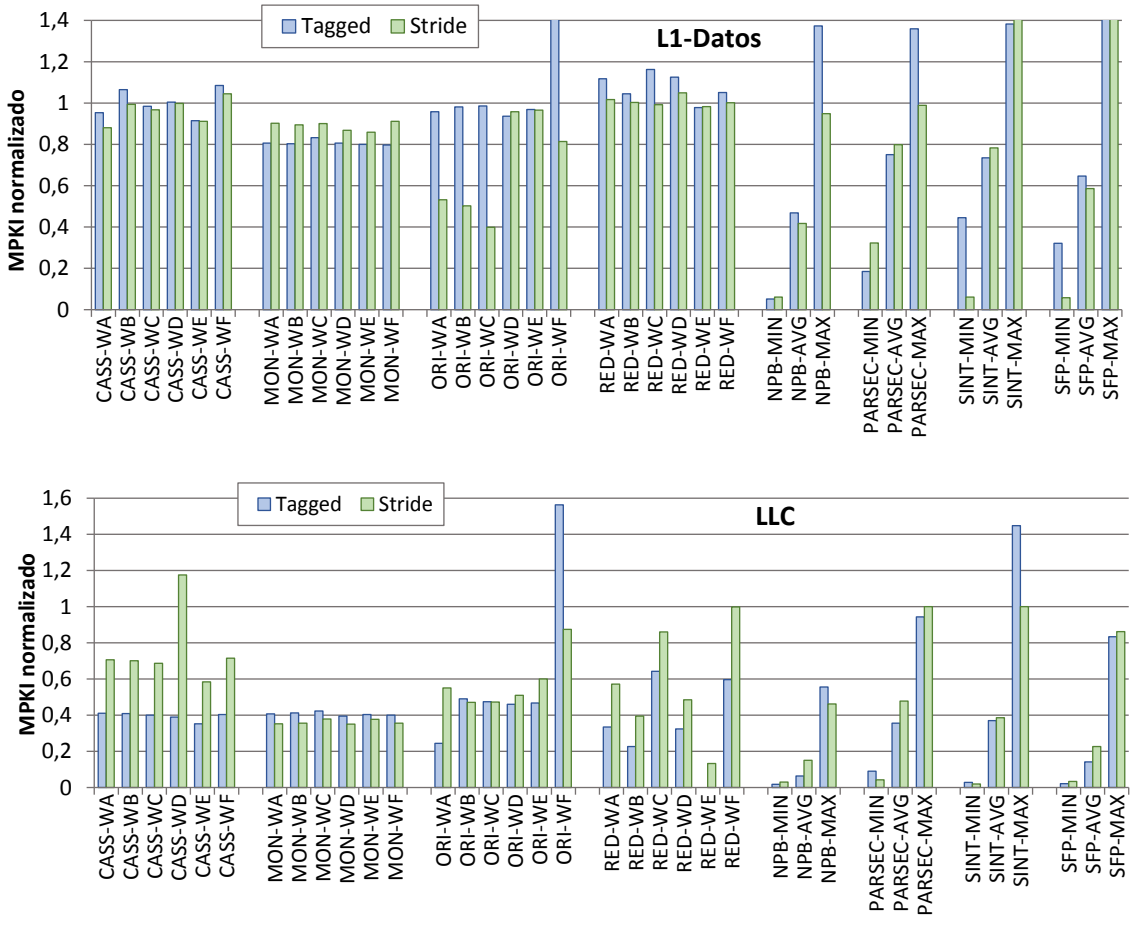

Figura 46 – Efecto de prefetching hardware en el MPKI. Los resultados están normalizados a los valores obtenidos en ausencia de mecanismos de prefetching.

4.3.4 Compartición de Bloques

Aplicaciones como Cassandra ejecutan, de manera simultánea, un número de *threads* elevado, cada uno encargado de diferentes etapas de ejecución, siendo, por tanto, aconsejable, completar la caracterización del comportamiento de las bases de datos NoSQL en la jerarquía de memoria con el análisis de la compartición de los bloques de memoria. Para la realización de este experimento ha sido necesario llevar a cabo las modificaciones necesarias, en la herramienta de

simulación, para permitir llevar la cuenta de cuántos *cores* han utilizado un determinado bloque desde que entra en la jerarquía de memoria hasta que es expulsado de vuelta a memoria principal. Los bloques se clasifican en "privados" o "compartidos", donde el grado de compartición varía entre 2 y 4 compartidores en el último caso, y las estadísticas se actualizan cuando un bloque es expulsado de la LLC. En este experimento se prescinde de las aplicaciones de SPEC puesto que son *single-thread*. Los resultados obtenidos se muestran en la Figura 47, donde se proporcionan datos sobre la fracción de bloques privados y compartidos, así como el número de compartidores (separados en instrucciones y datos).

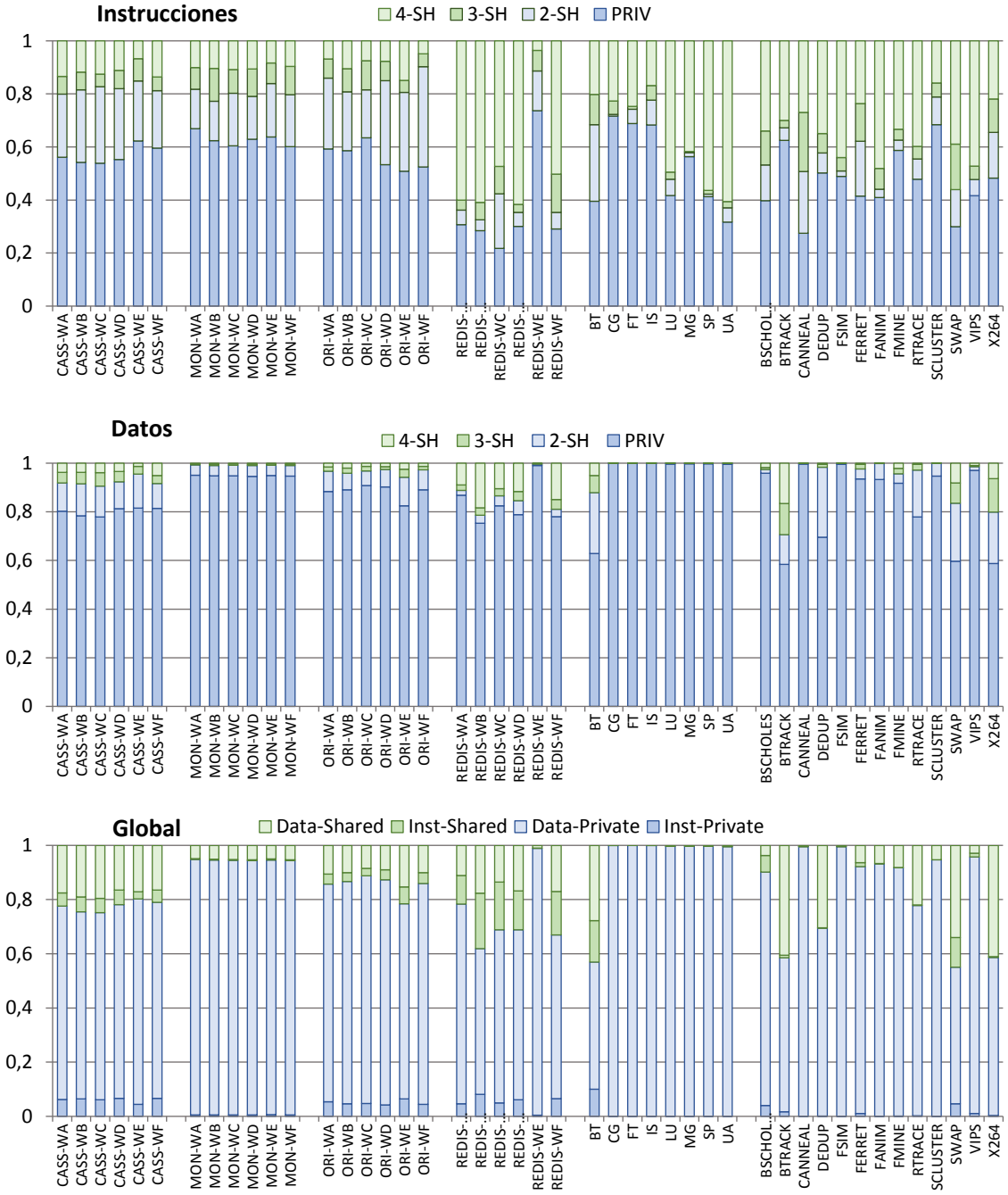

Figura 47 – Grado de compartición de instrucciones (arriba), datos (medio) y global (abajo) en el último nivel de la jerarquía de cache. El eje Y representa el porcentaje de compartición.

El análisis del grado de compartición revela un resultado común a ambos tipos de aplicaciones, el mayor grado de compartición del código con respecto a los datos. En la mayoría de las medidas, el porcentaje de bloques de instrucciones compartidos (referenciado por al menos dos *cores*) se sitúa en torno al 40%, mientras que esta cifra decrece notablemente para los datos. En lo concerniente a la compartición del código, los resultados de la base de datos Redis son similares a los de las aplicaciones convencionales, con un alto grado de compartición y donde la fracción de bloques compartidos está dominada por bloques que han sido referenciados por todos los *cores*. Cabe destacar que, al igual que sucedía en el tamaño del *working set* de instrucciones, los resultados de Redis en la ejecución del *workload* E son claramente distintos al resto de *workloads*, reduciéndose, en este caso, drásticamente, el grado de compartición. El resto de base de datos se sitúa en un plano distinto, exhibiendo un grado de compartición menor y cuya distribución del número de compartidores es más uniforme, si bien los bloques referenciados por dos *cores* son los que más predominan. Por otra parte, los resultados de la compartición de los datos de las aplicaciones NoSQL son más heterogéneos, tanto en el porcentaje de compartición como en la distribución del número de compartidores, siendo Cassandra y Redis las bases de datos con mayor grado de compartición. En cuanto a las aplicaciones convencionales, los resultados revelan que la compartición de datos en NPB es prácticamente inexistente y que en PARSEC el grado de compartición se reduce drásticamente en comparación con las instrucciones, de modo que las aplicaciones NoSQL presentan en general un mayor grado de compartición de los datos. Por último, el análisis conjunto (instrucciones y datos) del grado de compartición revela que las aplicaciones NoSQL presentan un grado de compartición igual o superior al de las aplicaciones convencionales, de modo que la eficiencia del protocolo de coherencia puede tener un mayor impacto en las aplicaciones NoSQL.

4.4 Conclusiones

En este capítulo se ha llevado a cabo de una detallada caracterización de aplicaciones emergentes utilizando una metodología de simulación propuesta en el capítulo III. El uso de dicha metodología proporciona un nivel de precisión y detalle en los resultados inalcanzable con metodologías alternativas, como el uso de contadores hardware o la utilización de herramientas de simulación con menor nivel de detalle. Se ha analizado un conjunto representativo de bases de datos de tipo NoSQL, comparando su comportamiento en la jerarquía de memoria con el de aplicaciones "convencionales", utilizadas habitualmente para la evaluación en el diseño de la micro-arquitectura del procesador. Los resultados obtenidos sugieren que el comportamiento en memoria de las aplicaciones NoSQL es similar al de las aplicaciones tradicionales en varios aspectos. Se aprecia la presencia de localidad espacial y temporal en datos e instrucciones, demostrada por los beneficios en rendimiento de mecanismos como la política de reemplazo o el *hardware prefetching*. De la misma forma, la evaluación también ha sido capaz de detectar sutiles diferencias en este tipo de aplicaciones, como el tamaño del *working set* de instrucciones, las peculiaridades del *working set* de datos o la fracción de *stores*. Cabe señalar, como resultado más destacable, la sorprendente uniformidad observada entre las aplicaciones NoSQL, revelando un comportamiento muy similar en diversos aspectos a pesar de tratarse de herramientas software completamente distintas. Ejemplos claros de este fenómeno son la independencia del tamaño de la base de datos o de las políticas de acceso en el rendimiento de la *cache* o la homogeneidad de la fracción de *stores* con independencia de la base de datos y del *workload*. Posiblemente, compartir una parte del complejo *stack* software (todas utilizan la misma JVM *(Java Virtual machine)*) está enmascarando las particularidades de cada base de datos con respecto a la arquitectura subyacente. Adicionalmente, se ha apreciado una significativa uniformidad en los

resultados de los distintos *workloads* analizados a pesar de la diferente mezcla de operaciones que utilizan, un resultado que se ha repetido en todas las bases de datos con las que se ha trabajado.

Los capítulos 3 y 4 han permitido obtener una metodología y un conjunto de cargas de trabajo apropiados para llevar a cabo propuestas con un entorno de evaluación consistente. La caracterización de las bases de datos NoSQL ha mostrado una respuesta positiva a gran parte de los mecanismos de rendimiento convencionales incluidos en jerarquías de *cache* actuales.

La última parte del trabajo realizado en esta tesis se ha dedicado al estudio y evaluación de tecnologías de memoria no volátil como reemplazo de los actuales transistores CMOS utilizados. La motivación de este trabajo se fundamenta, por una parte, en las crecientes necesidades, en términos de capacidad, de la jerarquía de memoria *on-chip* debido al progresivo aumento del número de *cores* y a las mayores demandas de las aplicaciones. En esta línea, los resultados obtenidos en el presente capítulo sugieren que tanto las aplicaciones convencionales como las bases de datos NoSQL se benefician del aumento de la capacidad debido a la presencia de localidad en el acceso a los datos. Por otra parte, los desafíos a los que se enfrenta actualmente la tecnología CMOS, en lo concerniente a la escalabilidad y energía, motivan la exploración de tecnologías alternativas para seguir escalando el rendimiento de la jerarquía de memoria. Estas tecnologías emergentes no están exentas de desafíos, de modo que, se propondrán soluciones arquitecturales que permitan hacer frente a las limitaciones tecnológicas que presentan y aprovechar al máximo todas sus ventajas de funcionamiento, como la mayor densidad de integración o el menor consumo energético.

5 Conjugando Tecnologías y Aplicaciones emergentes, propuesta basada en RMs

5.1 Introducción

Los crecientes desafíos en el proceso de fabricación de los transistores CMOS derivados de su paulatina miniaturización, han propiciado la exploración de múltiples alternativas para continuar diseñando y construyendo procesadores que permitan alcanzar mayores cotas de rendimiento. A pesar de que futuras evoluciones de la tecnología de fabricación con CMOS permitirán mantener un rendimiento creciente en un futuro cercano [144] (se prevé que al menos hasta el final de la siguiente década), llegará un momento en el que se hará necesario virar hacia vías alternativas para seguir escalando el rendimiento del hardware. Técnicas complementarias en el proceso de fabricación (*3D-stacking*), tecnologías disruptivas (memorias no volátiles o fotónica) o modelos de computación alternativos (*resistive computing*, *neuromorphic computing* o *quantum computing* [145]) son algunas de las propuestas actuales para superar el agotamiento del proceso de fabricación CMOS. Dichas propuestas se encuentran actualmente en diferentes grados de desarrollo, siendo las más maduras las correspondientes a modificaciones en el proceso de fabricación (*3D-stacking*) o determinadas tecnologías de memoria no volátil. En ambos casos, existen actualmente productos comerciales basados en dichas tecnologías [46][47][40][146][147] (fuera del ámbito del procesador por el momento).

La jerarquía de memoria es un elemento crucial en cualquier sistema de computación actual y debe escalar en términos de capacidad, eficiencia y coste para satisfacer las cada vez mayores demandas de las aplicaciones y del mayor número de *cores* presentes en los procesadores, evitando convertirse en un cuello de botella para el rendimiento [148][149]. Convencionalmente, la jerarquía de memoria presenta diferentes tecnologías en sus distintos niveles: celdas SRAM de seis transistores en los niveles embebidos dentro del chip, memoria principal construida con tecnología DRAM y dispositivos de almacenamiento con tecnología magnética o NAND-flash (disco duro de estado sólido o SSD). Durante décadas el continuo avance de la integración CMOS ha permitido a las tecnologías DRAM y SRAM aumentar la velocidad y la densidad de integración a un menor coste por bit. Sin embargo, se está llegando progresivamente a los límites físicos de escalabilidad que impedirán continuar miniaturizando ambas tecnologías, debido a condicionantes críticos como el creciente consumo de potencia, asociado a las corrientes de pérdidas de los transistores (*leakage*) o al refresco de datos en el caso de DRAM [150]. Las dificultades para seguir escalando el rendimiento de la jerarquía de memoria se agravan si tenemos en cuenta la creciente presión sobre capacidad y ancho de banda ejercida por las aplicaciones. El número de *cores* presente en cada procesador no deja de aumentar, con un *working set* de datos e instrucciones agregado más grande. Las aplicaciones emergentes de *Big Data* y los entornos de servidor (*Cloud Computing*), donde la consolidación es un aspecto clave, hacen un uso intensivo de la jerarquía de memoria, compartiendo los recursos de la memoria principal entre un alto número de aplicaciones/máquinas virtuales. En lo concerniente al procesador, la creciente demanda de memoria tiene como consecuencia la implementación de una jerarquía embebida con múltiples niveles, ocupando una porción muy significativa del chip y contribuyendo notablemente al consumo [51]. Con estos condicionantes, el interés en tecnologías alternativas que permitan seguir escalando la jerarquía de memoria se ha disparado en los últimos años, como demuestra el amplio conjunto de tecnologías propuesto [151].

Las tecnologías de memoria resistiva, a diferencia de las tecnologías DRAM o SRAM (las cuales se basan en el almacenado de carga eléctrica a través de un condensador o de inversores en un lazo cerrado respectivamente), hacen uso de diferentes fenómenos físicos para variar su valor de resistencia y así poder representar los valores lógicos 0 y 1. Ejemplos de este fenómeno de resistencia variable son la conmutación resistiva [152] o el cambio de magnetoresistencia [153], entre otros. Estas novedosas tecnologías presentan importantes ventajas con respecto a las actuales, como su densidad de integración [49] (reducido tamaño de celda) y sus ajustados consumos energéticos [148] (corrientes de pérdidas mínimas que reducen el consumo de energía estática). Sin embargo, no están exentas de desafíos que han limitado, por el momento, su adopción en todos los niveles de la jerarquía de memoria. Por una parte, el proceso de modificación del valor de resistencia (operación de escritura) parece presentar valores de latencia y consumo energético mayores que los de las tecnologías tradicionales [49]. Por otra parte, los valores de durabilidad (*endurance*) son, en general, varios órdenes de magnitud menores que los de las tecnología actuales [148]. En la jerarquía de memoria *on-chip*, donde la latencia es crítica y los accesos extremadamente frecuentes, la inclusión directa de dichas tecnologías debe buscar formas alternativas de superar las limitaciones mencionadas, siendo las soluciones micro-arquitecturales una de las herramientas que pueden dar respuesta a los desafíos existentes. El desarrollo de las tecnologías para alcanzar un punto mayor de madurez o el trabajo de los arquitectos de computadores será crucial para que en un futuro cercano se puedan postular con fuerza como sustitutas de las tecnologías DRAM y SRAM.

La parte final del trabajo desarrollado en esta tesis se centra en la evaluación arquitectural de tecnologías de memoria no volátil. Actualmente, estas tecnologías se sitúan en puntos distintos en cuanto a su grado de madurez, siendo ReRAM [154], PCM [155] y STT-RAM [44] sus máximos exponentes, pues cuentan todas ellas con productos comerciales o prototipos que demuestran su viabilidad [46][156][157][158]. De este conjunto, la tecnología STT-RAM, se alza como un serio candidato, debido a las características que presenta, no solo para la sustitución de la DRAM en la memoria principal, sino para ser integrada dentro del chip sustituyendo a SRAM. Sus valores de *endurance* o sus tiempos de acceso se acercan notablemente a los valores de SRAM [148] y, al mismo tiempo, presenta un tamaño de celda y un *leakage* asociado que mejora tanto a DRAM como a SRAM [148]. En la siguiente sección, se hace una descripción de las principales características de esta prometedora tecnología.

5.2 STT-RAM

La tecnología de memoria no volátil STT-RAM (*Spin-transfer torque RAM*) basa su funcionamiento en la utilización de elementos magnéticos y la propiedad de magnetorresistencia de estos materiales. El almacenamiento y acceso a los datos se implementa mediante lo que se denomina MTJ (*Magnetic Tunnel Junction*) [159] (ilustrada en la Figura 48) para básicamente variar la resistencia de la celda al paso de la corriente. Cada MTJ está formada por dos capas compuestas por un material ferromagnético, separadas por una capa metálica no magnética (por ejemplo, óxido de magnesio (MgO)) con un grosor muy fino. Una de las capas ferromagnéticas (denominada “capa de referencia”) tiene una dirección magnética fija, mientras que la otra capa (denominada “capa libre”) puede cambiar su dirección magnética haciendo circular una corriente polarizada a través de la MTJ [44]. La orientación magnética de la capa libre con respecto a la de la capa de referencia es lo que determina la resistencia de la celda al paso de la corriente, de modo que, si ambas capas tienen diferente dirección magnética, el valor de la resistencia de la MTJ es alto (representando un 1 lógico), mientras que si la dirección es la misma entonces la resistencia es

baja (representando un 0 lógico). La aplicación de un voltaje positivo en el punto B de la Figura 48 permite la escritura del valor 0, mientras que, si se aplica un voltaje negativo, se escribe el valor 1. El mantenimiento de la dirección magnética de la capa libre no necesita corriente, por lo que el *leakage* intrínseco de esta tecnología es prácticamente 0. Por otra parte, las operaciones de lectura se implementan aplicando un pequeño voltaje en los extremos de la MTJ. El valor de la corriente que circula es relativo a la resistencia de la MTJ y al compararlo con un valor de referencia es posible discernir el valor almacenado en la MTJ.

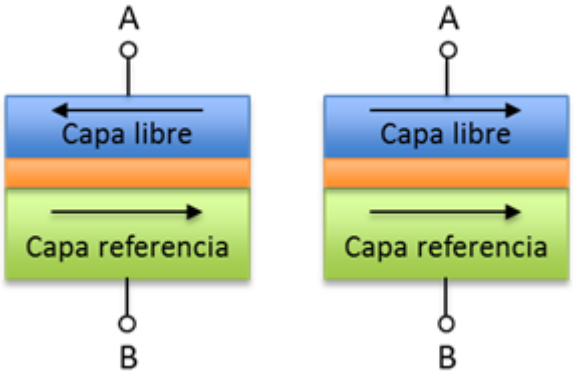

Figura 48 – Estructura de la MTJ. (izquierda) Alta resistencia (valor 1). (derecha) Baja resistencia (valor 0).

Actualmente, se estima que el valor del *endurance* de esta tecnología asciende a 10^{15} ciclos de escritura [49][160], siendo un valor prometedor para su integración dentro del chip (ya sea en todos los niveles de la jerarquía o únicamente en los más bajos) [161][162] y como sustituto de la tecnología DRAM en la memoria principal [163].

Su comercialización está liderada por Everspin, una compañía que tiene en el mercado desde hace tiempo chips de memoria (bajo la denominación STT-MRAM), fabricados en 40nm y con una capacidad de 256MB [46] y que se encuentra actualmente desarrollando una evolución del producto que alcanza un tamaño de 1GB en 28nm [164]. Adicionalmente, otras compañías entre las que destacan IBM y Samsung se encuentran también inmersas en el desarrollo comercial de esta tecnología, habiendo demostrado su escalabilidad hasta los 11nm [165].

El nivel de desarrollo de la tecnología STT-RAM, junto con las prometedoras características que potencialmente habilitarían su integración dentro del chip, han propiciado el desarrollo reciente de tecnologías alternativas basadas en los mismos principios denominadas *Racetrack Memories* [48]. Dicha evolución extiende significativamente la densidad de integración de STT-RAM mediante la sustitución de la capa libre por un nano-cable que posibilita el almacenamiento de múltiples bits por celda. En la última parte de esta tesis se va a trabajar con esta tecnología, a través de propuestas arquitecturales capaces de mitigar una de sus principales debilidades, la latencia variable de acceso a los datos.

5.3 Racetrack Memories

Las tecnologías de memoria denominadas *Racetrack Memories* (RMs) [48] son una evolución de la tecnología STT-RAM capaz de aumentar notablemente la densidad de bits por celda. La primera implementación de esta tecnología, denominada *Domain Wall Memory* o DWM [48], utiliza dominios magnéticos en un material ferromagnético para el almacenamiento de la información (ver Figura 49 (arriba)). Recientemente, se ha propuesto una tecnología similar, basada en la utilización de configuraciones magnéticas topológicamente estables denominadas *skyrmions*

[166] para la codificación de la información y denominada SK-RM (*Skymions Racetrack Memory*) (Figura 49 (abajo)), que consigue mayor densidad de integración, menor energía y mayor estabilidad de los datos que la DWM [167]. La gran ventaja de las RMs, como evolución de STT-RAM, reside en el aumento de la densidad de bits por celda, lográndolo mediante la utilización de un nano-cable que permite el almacenamiento de un número elevado de bits [48], siendo la longitud del cable lo que determina la capacidad de la celda. Las notables características de estas tecnologías, habiéndose demostrado la viabilidad de la integración de la tecnología DWM en el proceso de fabricación CMOS de 90nm [168], parecen indicar que son prometedoras de cara a futuro, motivo por el que se han escogido para el trabajo en esta tesis.

Ambas tecnologías se basan en la utilización de las propiedades magnéticas de materiales ferromagnéticos para modificar su valor de resistencia al paso de la corriente (ya sea mediante la utilización de dominios magnéticos o de *skyrmions*), logrando así el almacenamiento de los valores lógicos 0 y 1. El puerto de acceso a los datos, al igual que en la tecnología STT-RAM, es una MTJ, la cual permite realizar las operaciones de escritura mediante la aplicación de un determinado voltaje para inducir una determinada dirección magnética o para la creación de un *skyrmion*. Es necesario señalar, que en el caso de la RM basada en *skyrmions*, la destrucción de los mismos se implementa desplazándolos transversalmente al cable, lo que implicaría la adicción de un puerto extra para realizar esta operación [169]. De manera similar, las operaciones de lectura consisten en la aplicación de un determinado voltaje para comparar la corriente que circula a través de la MTJ [159] con un valor de referencia.

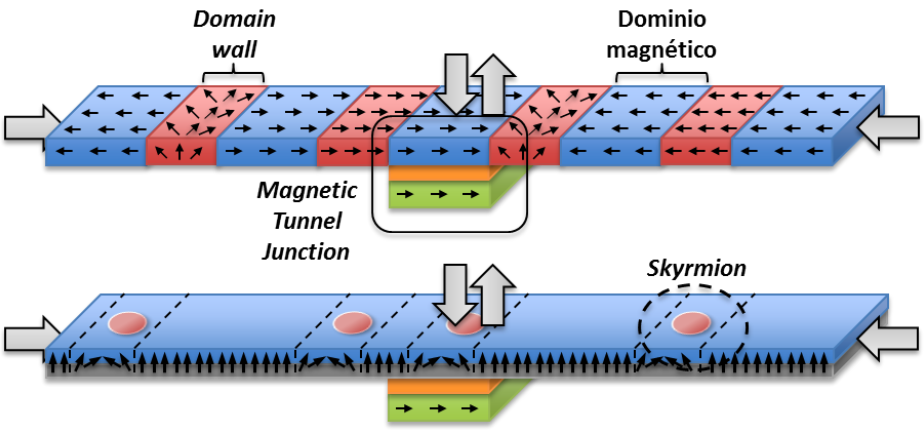

Figura 49 – Racetrack memories: DWM (arriba) y SK-RM (abajo).

Dado que el puerto de acceso a los datos (MTJ) es compartido por múltiples bits almacenados en la celda, el acceso a un determinado bit requiere alinear la porción del cable correspondiente con la MTJ antes de poder realizar la operación de lectura/escritura correspondiente. Esta operación de alineamiento se logra realizando múltiples desplazamientos (o *shitfs*) de los dominios magnéticos (o de los *skyrmions*) sobre el cable mediante la aplicación de un pulso de corriente entre sus extremos (ver Figura 49)**.** El flujo de corriente eléctrica provoca que la codificación de los bits se desplace en una determinada dirección, dentro del cable, debido a la transferencia del *spin* (o momento angular) y a la conservación de éste entre los electrones de la corriente y los átomos de material utilizado en el cable (debido al efecto denominado *spin-transfer torque* [170][171]) [48][167]. Con el objetivo de mejorar la eficiencia de las operaciones de desplazamiento y acercar la integración de estas tecnologías en el chip, se ha propuesto un método alternativo que logra reducir la energía y tiempo asociados a estas operaciones basado en el efecto *spin-orbit torque* [172]. Conlleva la incorporación al cable de una capa no magnética

(compuesta por un metal pesado) situada debajo del material ferromagnético, a través de la cual se hace circular una corriente eléctrica para inducir un cambio en la magnetización de la capa superior [173][167].

Esta peculiaridad implica que la latencia de los accesos en esta tecnología no es fija, sino que varía en función de la distancia existente entre el bit referenciado y la MTJ en el momento del acceso y, aunque es posible reducir la parte variable de la latencia mediante la adición de múltiples MTJs (espaciándolas a lo largo del cable), conlleva una importante penalización en área [51]. Resulta obvio el *trade-off* existente entre la latencia y la capacidad dado que, cuantos más bits es posible almacenar en el cable, mayor es la latencia media (y de peor caso) de los accesos, y viceversa. Además, para no perder información por los extremos, al realizar las operaciones de desplazamiento, es necesario que el cable esté correctamente dimensionado (en el caso de una única MTJ en la celda es necesario que el número de dominios (o *skyrmions*) que es posible inducir en el nano-cable sea igual al doble del número de bits que se desea almacenar). El compromiso existente, en esta tecnología de memoria, entre la capacidad y la latencia es lo que motiva este trabajo, enmarcándose éste en un escenario en el que se busca aprovechar los beneficios de una alta densidad de integración, utilizando, en cada celda, un alto número de bits (64) y un único puerto de acceso a los datos.

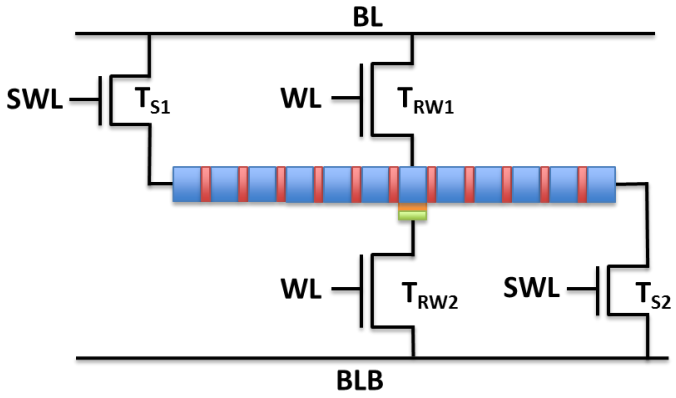

Figura 50 – Ejemplo de la implementación de una celda.

La estructura de celda necesaria para realizar las operaciones de lectura/escritura y desplazamiento de los bits para la tecnología DWM está representada en la Figura 50. Consta de un nano-cable, donde se almacenan los bits, una MTJ, y 2 parejas de transistores para realizar las operaciones (2 para las lecturas/escrituras y 2 para los desplazamientos). El área está determinada por el tamaño de los transistores, en especial por la pareja de la MTJ debido al mayor flujo de corriente de las operaciones de escritura. De este modo, es posible la utilización de celdas con un elevado número de bits, donde, según las estimaciones, el área por bit se situaría aproximadamente en un orden de magnitud menor con respecto a la tecnología SRAM [160]. Los valores de voltaje para cada una de las operaciones se encuentran resumidos en la Tabla 7. Las lecturas/escrituras del bit alineado con la MTJ se realizan precargando las *bitlines* (BL y BLB) con los valores apropiados y activando los transistores de acceso a la MTJ (T_{RW1} y T_{RW2}) para hacer circular la corriente a través de ésta. Las operaciones de lectura no son destructivas debido a que se utiliza un voltaje bajo, de manera que no se modifica la dirección magnética del dominio, al contrario de lo que sucede en las operaciones de escritura al utilizar voltajes más elevados. El flujo de corriente que circula a través de la MTJ varía en función de la resistencia al paso de la corriente, lo que a su vez está determinado por la dirección magnética del dominio y de la capa de referencia,

de modo que, si ambas direcciones son iguales entonces la resistencia es baja (valor lógico 0), mientras que en caso contrario el valor de la resistencia es alto (valor lógico 1) [160]. En el caso de la SK-RM, la presencia de un *skyrmion* en el nano-cable provoca que la resistencia sea baja (bit 1) y viceversa [174]. Comparando el flujo de corriente con un valor de referencia es posible determinar el valor lógico del bit almacenado en una determinada porción del cable. En el caso de las operaciones de escritura, el procedimiento es idéntico al de las operaciones de lectura, pero utilizando un voltaje mayor que consigue programar la dirección magnética del dominio alineado con el puerto de acceso en una determinada dirección (o la creación de un *skyrmion*). Por último, las operaciones de desplazamiento hacen uso de la pareja adicional de transistores incluida en la celda (T_{S1} y T_{S2}) y que está conectada a los extremos del cable. En lo concerniente a la SK-RM, la celda tendría una implementación muy similar puesto que las operaciones de desplazamiento, las lecturas y las escrituras del valor lógico 1 (creación de un *skyrmion*), se implementarían a través de la MTJ. Sin embargo, la escritura del valor lógico 0 en presencia del valor 1 en el cable (destrucción del *skyrmion*) requeriría la adición de un puerto extra [169].

Tabla 7 – Valores de tensión de las distintas líneas para cada una de las operaciones.

	BL	BLB	WL	SWL
Read	V_{read}	0	V_{DD}	0
Write 0	V_{DD}	0	V_{DD}	0
Write 1	0	V_{DD}	V_{DD}	0
Shift left	V_{DD}	0	0	V_{DD}
Shift right	0	V_{DD}	0	V_{DD}

Al margen de las múltiples propuestas arquitectónicas basadas en esta tecnología (principalmente enfocadas a la tecnología DWM), se han hecho progresos notables desde su definición en la pasada década en posibles puntos débiles como son la escalabilidad (manteniendo la estabilidad térmica) o la energía necesaria para los desplazamientos, tal y como demuestra un prototipo fabricado en 20 nm [175]. En [176] se propone la fabricación del cable y la MTJ con un material (CoFeb) para el que se ha demostrado la existencia de PMA (*Perpendicular Magnetic Anisotropy*), de modo que los *spins* pueden ser magnetizados perpendicularmente al plano. Los autores afirman que este material es prometedor para mejorar aún más las notables características de la DWM en términos de densidad, velocidad y consumo. En la misma línea, los autores de [177] y [178] exploran métodos alternativos para el movimiento controlado de los dominios en el cable en materiales con alta PMA. En los primeros diseños de DWM se proponía el uso de muescas en el cable para tal fin que aumentaban su resistencia y por tanto la corriente necesaria para las operaciones de desplazamiento. Sin embargo, la utilización de materiales con alta PMA permite modificar la velocidad del movimiento de los dominios mediante campos eléctricos perpendiculares al cable para así controlar su desplazamiento. En [179] se explora el uso de campos magnéticos para reducir la corriente necesaria en el desplazamiento de los dominios a lo largo de cable. Adicionalmente, los autores de [180] y [181] proponen la utilización de esta tecnología disponiendo el cable en forma de anillo. Con este diseño, se evitaría tener que dimensionar el cable para el almacenamiento de bits adicionales con el fin de evitar la pérdida de información por los extremos y, además, se reducirían las operaciones de desplazamiento.

En lo concerniente a la SK-RM, el interés desde el ámbito de la electrónica es más reciente que en el caso de la DWM, si bien existen trabajos recientes que tratan de mejorar esta tecnología con vistas a su implementación. Así, en [182] y [183] se propone la utilización de dos nano-cables para el almacenamiento de los *skyrmions* ante los problemas que puede acarrear la manipulación de largas secuencias de bits con un valor “0”. La representación de este valor lógico mediante la

ausencia de *skyrmions* en el cable dificulta la determinación del número de bits, debido a que la distancia entre las posiciones de los bits en el cable puede, en la práctica, ser variable. Mediante la utilización de dos nano-cables, mapeando cada uno de los dos valores lógicos a un cable concreto, se facilita la manipulación de los bits. Otros trabajos ponen el foco en la problemática existente en el movimiento de los *skyrmions* en el nano-cable, ya que éstos pueden desviarse del flujo de corriente y destruirse al chocar con los bordes de nano-cable. Así, los autores de [184] proponen la utilización de materiales con alta anisotropía magnetocristalina en los bordes del nano-cable para confinar a los *skyrmions* en el centro de éste de manera eficiente e impedir su destrucción. En la misma línea, en [185] se proponen dos soluciones alternativas para el mismo problema. La primera consiste en la modificación de la anisotropía magnética del cable a través de la radiación de iones, logrando reducir la anisotropía en el centro con respecto a los bordes. De este modo, se crea un camino de baja resistencia en el cable con respecto a los ejes que evita que los *skyrmions* se destruyan durante los desplazamientos. La segunda solución que proponen consiste en la modificación de la geometría del cable, creando una ranura rectangular en el centro de una capa adicional de material situada entre los bordes que evita que los *skyrmions* se desvíen hacia los bordes, impidiendo así su destrucción.

A pesar de los múltiples avances tecnológicos mencionados, uno de los principales condicionantes para la adopción de las RMs sigue estando en el carácter variable de la latencia en los procesos de lectura y escritura. Es en este ámbito, donde la Arquitectura de Computadores puede ser capaz de ofrecer soluciones capaces de paliar o minimizar el efecto del *overhead* en la latencia. Hasta la fecha, son varias las soluciones propuestas en la literatura, como diseños de *cache* reconfigurables que varían dinámicamente el compromiso entre capacidad o latencia, técnicas de compresión y migración de datos o políticas de reposicionamiento del puerto de acceso. En la misma línea, el trabajo que cierra esta tesis se ha centrado en este ámbito, proponiendo una política de reposicionamiento del puerto de acceso más genérica que las existentes, una solución sencilla que no conlleva una compleja manipulación de los datos como sucede en otras propuestas.

5.4 Soporte Arquitectural para RMs: Estado del Arte

Son múltiples los trabajos basados en las RMs elaborados en el ámbito de la Arquitectura de Computadores durante los últimos años, principalmente centrados en la tecnología DWM, pero dadas las similitudes entre la DWM y la SK-RM, la aplicación de estos trabajos sería, previsiblemente, extensible a ésta última. En concreto, se ha propuesto su utilización en los distintos niveles de la jerarquía de memoria, desde el banco de registros del procesador, pasando por la jerarquía de memoria *on-chip* y la memoria principal, hasta el almacenamiento en disco, incluyendo su uso en GPUs (memoria y registros). Los desafíos provocados principalmente por las operaciones de escritura y desplazamiento en términos de latencia y energía, así como la fiabilidad de los desplazamientos, han motivado una serie de propuestas a nivel arquitectural para tratar de solventarlos y postular a la tecnología RM como sustituta de SRAM en la jerarquía de memoria de los procesadores. Esta sección se limita a la revisión de las propuestas arquitectónicas más relevantes, relacionadas con la propuesta de este capítulo. Por simplicidad, en lo que resta de capítulo se unifica la nomenclatura, utilizando la palabra *dominio* para hacer referencia a la porción de nano cable que alberga un bit (dominio magnético (DWM) o un *skyrmion* (SK-RM)). De igual forma, se utiliza el término RM (en singular) para hacer referencia al conjunto de tecnologías similares que almacenan los bits en un nano-cable.

En lo concerniente a la energía y la latencia de las operaciones de escritura, los autores de [52] proponen la utilización del mismo principio empleado en el desplazamiento de los dominios a lo

largo del cable para su implementación, es decir, mediante la aplicación de un pulso de corriente eléctrica por uno de los extremos. Así, la capa libre de la MTJ (en este caso, una determinada porción del nano-cable) se rodea por sendos dominios magnéticos cuyas direcciones magnéticas son antiparalelas (y fijas, siempre y cuando se dimensionen apropiadamente) y que se sitúan transversalmente con el nano-cable (ver Figura 51). La aplicación de un pulso de corriente permite trasladar el bit codificado en cada uno de estos dominios adicionales hasta la capa libre de la MTJ y, por tanto, la escritura de los valores 0 y 1. En comparación con la implementación de la MTJ convencional, este diseño alternativo obtiene un menor tiempo y energía por cada operación de escritura y mejora la densidad de integración (el pulso de corriente es menor y por tanto los transistores de acceso son más pequeños) [52].

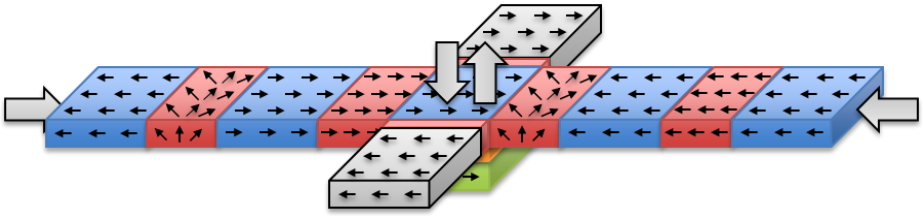

Figura 51 – Racetrack con operaciones de escritura basadas en desplazamientos.

Por otra parte, los problemas de fiabilidad presentes en la tecnología DWM están relacionados con las operaciones de desplazamiento [186] y las variaciones en el proceso de fabricación [187]. Los desplazamientos pueden provocar situaciones en las que un dominio no queda correctamente alineado con el puerto de acceso o en las que se ha producido un desfase de varios dominios. Para aliviar el primer problema, los autores de [186] proponen la realización de los desplazamientos en dos fases (utilizando sendos pulsos de corriente con distinta intensidad), consiguiendo reducir notablemente el mal alineamiento, pero al mismo tiempo incrementando los desfases. Para solucionar el segundo problema los mismos autores sugieren el uso de un código corrector de errores utilizando bits adicionales en los extremos de cable, consiguiendo corregir desfases de 1 posición y detectar desfases de 2 posiciones. Por otra parte, las variaciones en el proceso de fabricación de esta tecnología pueden provocar un aumento significativo en la latencia de las distintas operaciones, de modo que los autores de [187] proponen la aceleración de las operaciones en aquellos dominios que presenten variaciones, mediante un aumento de la intensidad de la corriente que acelere su tiempo de ejecución y minimice el impacto de este problema.

Adicionalmente, las dificultades derivadas de las operaciones de desplazamiento en lo referente a la latencia y energía asociadas han concentrado gran parte de los trabajos publicados. Así, una vía que han utilizado algunos autores para minimizar el problema de la latencia variable es la posibilidad de reconfiguración de la *cache* en base a la inhabilitación de dominios de las celdas. Los autores de [188] proponen una *cache* reconfigurable dinámicamente que busca alcanzar el equilibrio entre capacidad y latencia para así adecuarse a la distinta sensibilidad de las aplicaciones a ambos aspectos. El uso de la tecnología RM ofrece un mecanismo natural para reconfigurar la *cache* basado en la limitación del número de dominios activos, logrando así adecuar la *cache* en términos de capacidad y latencia de los accesos a los requerimientos de la aplicación. La decisión para aumentar o disminuir la capacidad de la *cache* se basa en la monitorización de una serie de factores como el *miss rate* o la *shift latency*. La reconfiguración de la *cache* implica un proceso de migración de datos al cambiar el mapeo de la misma, de modo que para minimizar el *overhead* asociado a este proceso se utiliza una política relajada, donde tras un período determinado los datos que aún residen en una posición incorrecta son migrados forzosamente a su posición

correcta o expulsados a memoria. Además, se propone la utilización de los dominios inhabilitados como una *victim cache* cuando la capacidad efectiva de la *cache* es menor que la capacidad máxima para así reducir el número de accesos a memoria principal. Un mecanismo similar ha sido propuesto en [189] para adecuarse, de igual forma, a la necesidad de capacidad de las aplicaciones. Utilizan un mapeo por vías, posibilitando la reconfiguración del tamaño de la *cache* en base al número de vías activas. La reducción del número de vías minimiza las operaciones de desplazamiento, consiguiendo mejorar el rendimiento y reducir la energía.

Otra propuesta se basa en un diseño de *cache* que utiliza un único almacenamiento físico para combinar dos niveles lógicos de la jerarquía: L1 y L2 [190]. En su diseño, los dominios que están alineados con los puertos de acceso forman parte de L1, puesto que su latencia es fija al no requerir operaciones de desplazamiento, mientras que el resto de los dominios forman parte de L2. Al utilizar un número de dominios reducido se amortigua el impacto de la latencia variable en el acceso a L2. El movimiento de bloques a nivel lógico se realiza mediante *shifts* y en determinados casos mediante *swaps* en *background*. Su diseño utiliza el algoritmo tradicional LRU para las expulsiones y permite que la asociatividad de L1 y L2 sea la misma o distinta.

Otras propuestas alternativas para mitigar la latencia y la energía asociadas a las operaciones de desplazamiento se basan en la migración y compresión de los datos. En [191] se propone la utilización de un mapeo por vías para así poder migrar los bloques más utilizados a los dominios alineados con los puertos. De este modo, se reduce el número de operaciones de desplazamiento necesarias y, en consecuencia, la latencia y la energía de los accesos. Por otra parte, los autores de [192] proponen reducir la energía de las operaciones de lectura/escritura mediante el uso de técnicas de compresión. Además, gracias a que la capacidad liberada por la compresión no se utiliza para almacenar información adicional, es posible reducir la granularidad de las operaciones de desplazamiento para reducir su energía. Adicionalmente, proponen no alinear la información comprimida con el inicio del bloque para poder realizar accesos simultáneos a distintos bloques de datos y aumentar el rendimiento.

Estos trabajos se centran en los datos como vía para la reducción de la latencia variable, ya sea a través de costosos y complejos procesos de migración o de técnicas de compresión, donde no se aclaran los potenciales efectos que pueden tener en aspectos claves como la coherencia o la consistencia. Sin embargo, algunos autores han explorado una vía alternativa con el mismo objetivo, consistente en el movimiento de la cabeza tras cada acceso en aras de reducir o eliminar las operaciones de desplazamiento del siguiente acceso. Las primeras propuestas se centran en políticas básicas de reposicionamiento, denominadas *Lazy* y *Eager* [51]. La primera trata de explotar la localidad de los accesos, dejando el puerto de acceso alineado con el último bit accedido esperando que la siguiente operación haga referencia a un bit cercano. La segunda trata de reducir la latencia de peor caso fijando una posición por defecto para cada puerto de acceso, de modo que tras cada lectura/escritura comienza a realizar sucesivas operaciones de desplazamiento para alinear los puertos con los dominios correspondientes. En el caso en que el siguiente acceso llegue antes de poder retornar a la posición fija, el dominio se encontrará en una posición comprendida entre el último dominio accedido y el dominio fijo. En [52] se propone una política que predice de manera fija que el siguiente bit que va a ser referenciado. En concreto, predice que el siguiente bit que se va a acceder está situado una posición a la derecha de último bit accedido, es decir, que se va a acceder al siguiente bloque de memoria, de modo que tras cada acceso desplaza la cabeza lectora apropiadamente. Además, se han propuesto mecanismos de predicción para GPUs más complejos, similares a los empleados para la especulación del flujo de control (predicción de saltos) en el *pipeline* [193][194]. Son mecanismos que se basan en la historia

pasada, de modo que necesitan tablas para ir progresivamente almacenando información sobre los últimos accesos realizados y poder así adivinar, en base a esa información, el siguiente dominio que será accedido. De manera similar, la propuesta realizada eneste trabajo también basa su funcionamiento en un mecanismo de reposicionamiento de la cabeza lectora. A diferencia de las descritas anteriormente, se propone el uso de un mecanismo similar a los utilizados para el *prefetch* hardware de bloques en la jerarquía de memoria para adivinar el siguiente bloque que será accedido y alinear el dominio correspondiente con el puerto de acceso.

5.5 RM *Preshifting*

La tecnología RM permite el almacenamiento de un elevado número de bits por celda, dimensionando el cable ferromagnético apropiadamente. El número de dominios que puede albergar el cable parece poder situarse por encima de la centena [195], de modo que sería factible la implementación de celdas que almacenasen hasta 64 bits (más los dominios adicionales necesarios para evitar la pérdida de información por los extremos en las operaciones de desplazamiento). Sin embargo, la implementación de celdas de alta densidad tiene como contrapartida que la latencia de acceso aumente debido a la necesidad de alinear el bit correspondiente con el puerto de lectura/escritura, siendo este *trade-off* entre la capacidad y la latencia uno de los principales desafíos de esta tecnología. Ante esta situación, la política de reposicionamiento se convierte en un elemento de vital importancia, puesto que es el único medio para sufragar, en la medida de lo posible, el *overhead* que conllevan los desplazamientos.

La precisión de algunas de estas políticas (*Lazy* y *Preshift*) depende de la localidad en el acceso a los datos, exhibiendo un buen grado de rendimiento en presencia de localidad [196], pero con un peor comportamiento cuando el grado de localidad disminuye. La localidad no es una característica constante, pudiendo variar con la aplicación o con el nivel de la jerarquía de *cache on-chip*. Para compensar esta heterogeneidad, se propone una política híbrida que combina un mecanismo de *preshifting* (o reposicionamiento), basado en el reconocimiento de patrones en los accesos a la *cache*, con *Lazy*, una política fuertemente centrada en explotar la localidad (se escoge la política con mejor comportamiento de las analizadas). En esta sección del capítulo se describen con detalle todos los aspectos de la política de reposicionamiento de la cabeza lectora que se propone. Se comienza explicando la organización básica de la cache (sección 5.5.1) para, a continuación, explicar el funcionamiento de la política, ilustrándolo con un ejemplo sencillo (sección 5.5.2). Posteriormente, se lleva a cabo una exploración de la sensibilidad de la política a los parámetros de configuración (sección 5.5.3), para maximizar el rendimiento y minimizar su *overhead* en área y energía. La configuración final, así como la estimación del coste de implementación se detallan en la sección 5.5.4. En la sección 5.5.5, se evalúa el rendimiento de la política propuesta comparándola con las tres políticas descritas en la sección anterior: *Lazy*, *Eager* y *Preshift*. Por último, se discute la integración de la tecnología RM como sustituta de SRAM en los niveles más bajos de la jerarquía cache *on-chip* (sección 5.5.6).

5.5.1 Organización de *Cache* Básica

A la hora de construir un banco de *cache* con este tipo de tecnología, la organización de los datos sobre las celdas puede realizarse mapeando los bits de un bloque secuencialmente sobre una o varias celdas de RM (ver Figura 52 (arriba)), donde un bloque de *cache* de longitud M bits, se almacena en una o varias celdas con N dominios, dependiendo del tamaño de bloque y del número de bits/dominios por celda. El inconveniente de esta aproximación radica en el acceso secuencial a los datos puesto que independientemente de la longitud del bloque, la utilización de celdas con

N bits implicaría la realización secuencial de N operaciones de lectura/escritura y N-1 desplazamientos para leer un bloque, incrementando significativamente la latencia de acceso a los datos. Un mapeo alternativo consiste en el intercalado de los bits de un bloque de memoria en múltiples celdas, en concreto de un bit por celda (ver Figura 52 (abajo)). De este modo, un bloque de longitud N se almacena en N celdas (1 bit por celda) y una celda con capacidad para M bits almacena 1 bit de M bloques distintos. Al utilizar el segundo mapeo, el conjunto de celdas funciona como una unidad, de manera que las operaciones de desplazamiento y lectura/escritura se realizan en paralelo y, en un instante dado, todas las celdas están alineadas con el mismo dominio, pudiendo leer un bloque completo en un solo ciclo. Sin embargo, existe contención en el acceso a los datos de modo que no es posible acceder más de un bloque al mismo tiempo.

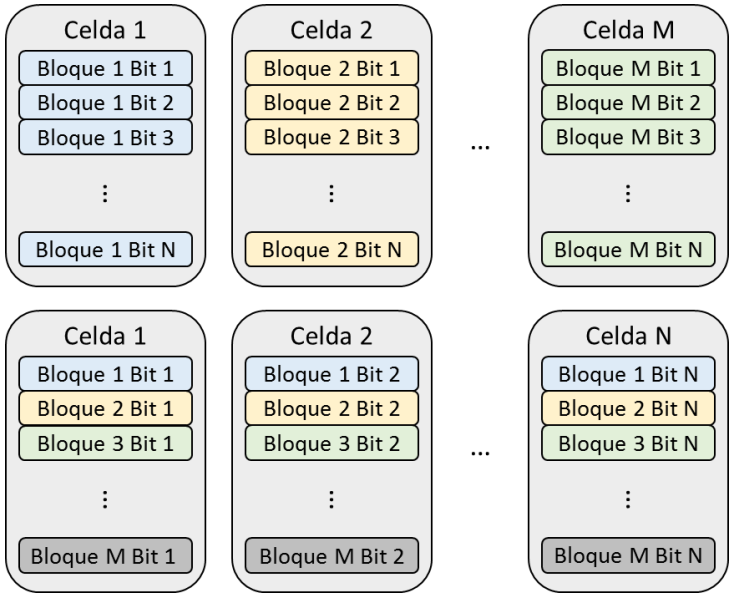

Figura 52 – Ejemplo de la organización de los datos. (arriba) Secuencial. (abajo) Intercalado.

En base al intercalado de los bits de un bloque en múltiples celdas es posible organizar los *sets* de la *cache* de dos maneras, posicionando una vía concreta de distintos *sets* secuencialmente a lo largo del cable o ubicando los *sets* completos de modo secuencial. La Figura 53 ilustra los dos modos de organizar los datos mencionados utilizando bloques cuya longitud es 64 bits, un valor de asociatividad 2 y celdas con 32 dominios. En la primera aproximación (Figura 53 (arriba)), cada una de las celdas necesarias (64 en este ejemplo) para el almacenamiento de un bloque de *cache* almacena un bit de una determinada vía de N *sets* distintos (siendo N el número de bits por celda). Por el contrario, en el segundo mapeo (Figura 53 (abajo)), las distintas vías (dos en este ejemplo) que forman los *sets* se almacenan de manera secuencial. El mapeado por vías presenta una gran desventaja frente la organización por *sets* con respecto a su utilización en *caches* con acceso paralelo, donde los datos y los *tags* se leen al mismo tiempo. Al ubicar todas las vías de manera consecutiva se penaliza en exceso el tiempo de acceso a los datos puesto que serializa la lectura de los bloques almacenados en el conjunto de vías.

Una posible organización de la *cache* utilizando esta tecnología está ilustrada en Figura 54. Consiste en un *array* de datos construido utilizando celdas RM con un mapeo en base a *sets*, un *array* para los *tags* (no mostrado) y la lógica necesaria para controlar la posición de la cabeza y la realización de las operaciones de desplazamiento. El *array* de *tags* se implementa con tecnología SRAM dado que el área que ocupa es mínima en comparación con los datos. La lógica de las filas controla las líneas *wordline*/*shiftline* que llegan a cada celda y que seleccionan el tipo de operación

a realizar activando los transistores pertinentes. Por otra parte, la lógica de las columnas gobierna las líneas BL y BLB para realizar las operaciones de lectura/escritura y desplazamiento. Los campos de la dirección *tag* y *offset* se utilizan como en una *cache* convencional, sin embargo, los bits del *index* se dividen en dos partes: los bits de la celda y los bits de los dominios. Los bits de la celda se utilizan para seleccionar una fila concreta de celdas RM, mientras que los bits del dominio se utilizan para seleccionar un dominio en particular de todas las celdas seleccionadas en la fila.

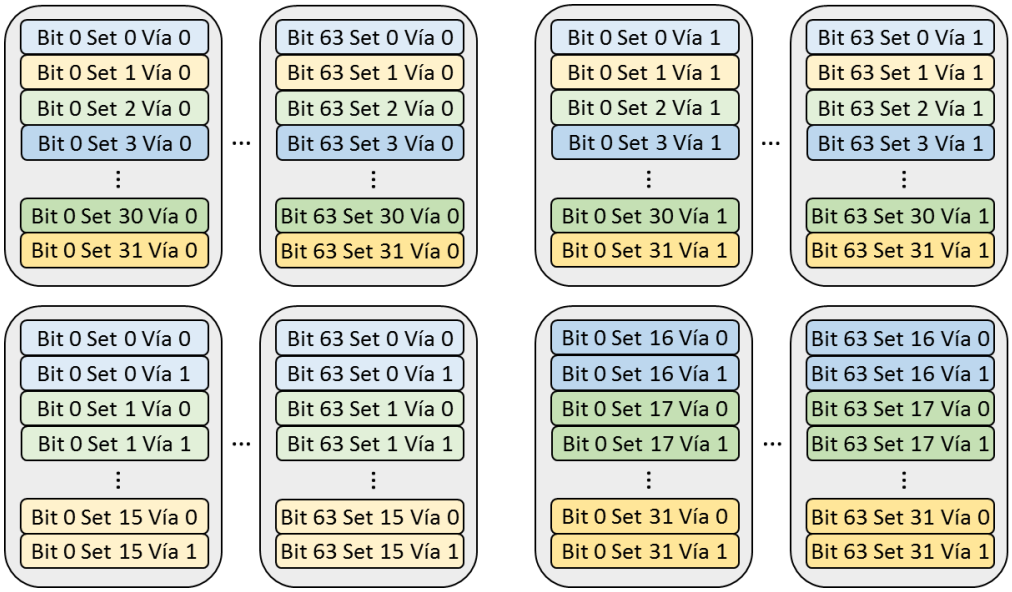

Figura 53 – Organización de los sets. Mapeo por sets (arriba). Mapeo por vías (abajo).

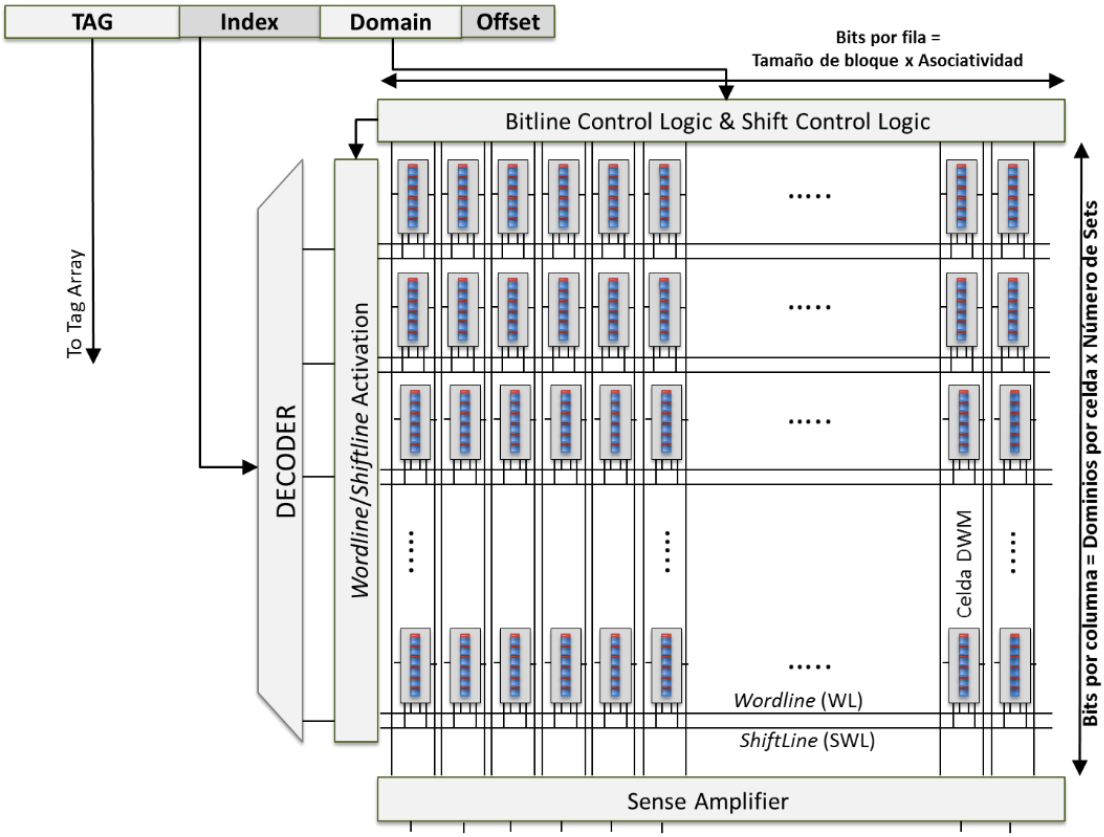

Figura 54 – Organización de la cache.

5.5.2 Estructura

El mecanismo para la identificación de los patrones en los accesos a los bits de las celdas se basa en una técnica de *prefetch* hardware correlativo [197][198], que trata de identificar patrones de acceso irregulares. Estos algoritmos necesitan almacenar la historia reciente para poder realizar las predicciones, ya sea almacenando las direcciones de las referencias que provocan un *miss* o la distancia relativa entre éstas [199]. Así, utilizan una tabla para almacenar, al menos, la relación entre dos *misses* consecutivos, de manera que, si se repite un *miss* de la primera referencia, se desencadena un *prefetch* para obtener el segundo bloque. A diferencia de los *prefetchers* de memoria, donde la utilidad de las predicciones tiene un cierto margen en el tiempo y puede haber, en un momento dado, varios bloques en la *cache* cuya presencia de debe a una petición de *prefetch*, en el *preshifting* del puerto de acceso de las celdas RM el margen es mucho más reducido. En concreto, únicamente interesa conocer la ubicación del bit que va a ser accedido por el siguiente acceso. La política que se propone emplea un mecanismo similar al descrito, donde la información que se utiliza para la definición e identificación de patrones es la distancia entre accesos consecutivos, es decir, el número de operaciones de desplazamiento. Esta distancia se obtiene siempre como la diferencia entre los dominios referenciados por dos accesos consecutivos. La información de los desplazamientos incluye la magnitud (distancia entre dominios en la celda) y la dirección (izquierda o derecha). Cabe señalar que existe la alternativa de trabajar con patrones compuestos por el identificador de dominio, pero la explicación de la propuesta se lleva a cabo utilizando patrones de distancia, puesto que ofrecen un mejor rendimiento como se verá posteriormente.

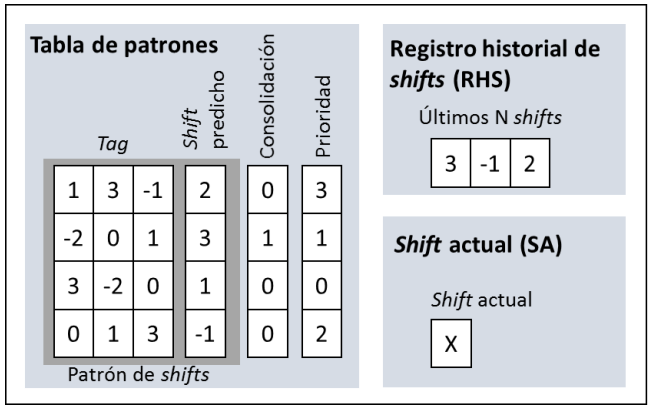

Figura 55 – Estructuras hardware utilizadas por la política propuesta.

El mecanismo que se propone hace uso de dos estructuras hardware, denominadas *Tabla de Patrones* (TP) y *Registro de Historial de Shifts* (RHS). Además, el *shift* asociado al acceso en curso está disponible a través un registro denominado *Shift Actual* (SA). Un esquema con las estructuras descritas (TP, RHS y SA) está representado en la Figura 55. EL RHS almacena los últimos N *shifts* ocurridos, mientras que la tabla de patrones almacena la historia pasada, es decir, un número determinado de patrones de desplazamiento pasados. Cada entrada de dicha tabla contiene los siguientes campos:

- Patrón de *shifts*: Contiene información sobre los últimos W *shifts* consecutivos (W = 4 en la Figura 55). Cada entrada está compuesta por el campo denominado "*Tag*" (con un valor [1 3 -1] en la primera fila de la tabla) y el "*Shift* predicho" (con un valor 2 en la primera fila de la tabla). El valor del RHS se compara con todos los valores *tag* de la tabla para realizar la predicción. El campo *shift predicho* es el valor utilizado para el reposicionamiento de la

cabeza (o *preshift*). La longitud del campo *tag* depende directamente de la longitud de los patrones, mientras que el campo *shift predicho* solo almacena un valor.

- Consolidación: Número de veces que se ha repetido el patrón correspondiente.
- Prioridad: Si el tamaño de la tabla no permite almacenar todos los posibles patrones, la política de reemplazo utiliza este campo para seleccionar el patrón que va a ser reemplazado por otro. En la figura, esta política imita el algoritmo LRU utilizado para el reemplazo de bloques en *caches* asociativas.

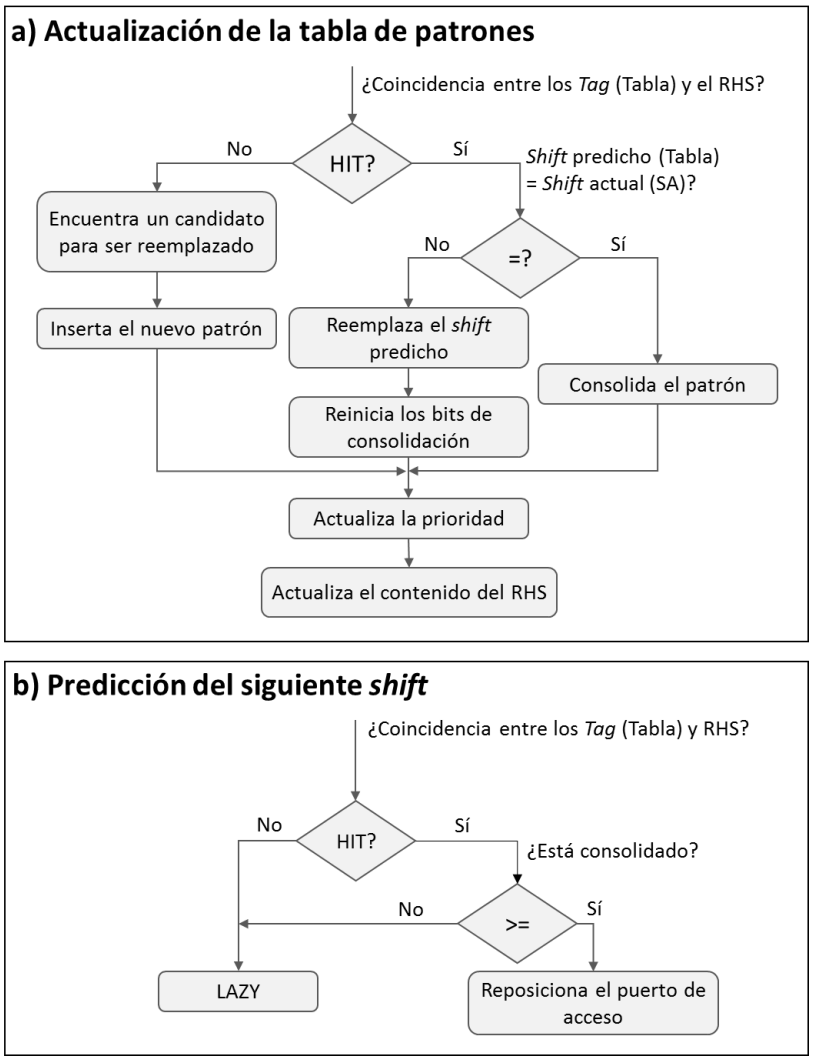

Figura 56 – Diagrama describiendo: (a) El proceso de actualización de los patrones de la tabla. (b) La predicción del siguiente shift.

En cada acceso a la *cache*, es necesario realizar dos operaciones de manera secuencial: la actualización del contenido de la tabla y la predicción (si procede) del próximo dominio que será accedido. La Figura 56.a ilustra los pasos involucrados en el proceso de actualización de la tabla de patrones después de cada acceso. Comienza buscando una entrada de la tabla cuyo campo *tag* sea igual al contenido del RHS. En caso de acierto, se compara el campo *shift predicho* de la entrada de la tabla correspondiente con el contenido del *shift actual*. Si ambos valores son idénticos, el valor del campo *consolidación* de la fila de la tabla de incrementa. En caso de que ambos valores difieran, se sustituye el valor del campo *shift predicho* por el valor del *shift actual* y, además, se reinicia el valor de *consolidación*. Si el patrón no se encuentra en la tabla debe

insertarse, siendo necesaria la expulsión de una determinada entrada mediante el uso del algoritmo semejante a LRU. Los últimos pasos de este proceso consisten en la actualización de los valores de prioridad de las entradas de la tabla y del RHS. El contenido del RHS se desplaza para insertar el valor del *shift actual* como parte de los últimos N *shifts*. A continuación, comienza el proceso para la predicción del siguiente dominio que será accedido, tal y como se describe en la Figura 56.b. El primer paso consiste en la búsqueda de una entrada en la tabla cuyo campo *tag* sea igual al contenido (ya actualizado) del RHS (tal y como tiene lugar en el proceso anterior). En caso de acierto y si el valor del campo *consolidación* de la fila correspondiente es superior a un determinado valor umbral, el puerto de acceso de reposiciona de acuerdo con el valor del campo *shift predicho*. En caso contrario, bien porque no se haya encontrado una coincidencia en la tabla o porque no se supere el valor umbral de consolidación, se aplica la política *Lazy*, es decir, no se reposiciona el puerto de acceso.

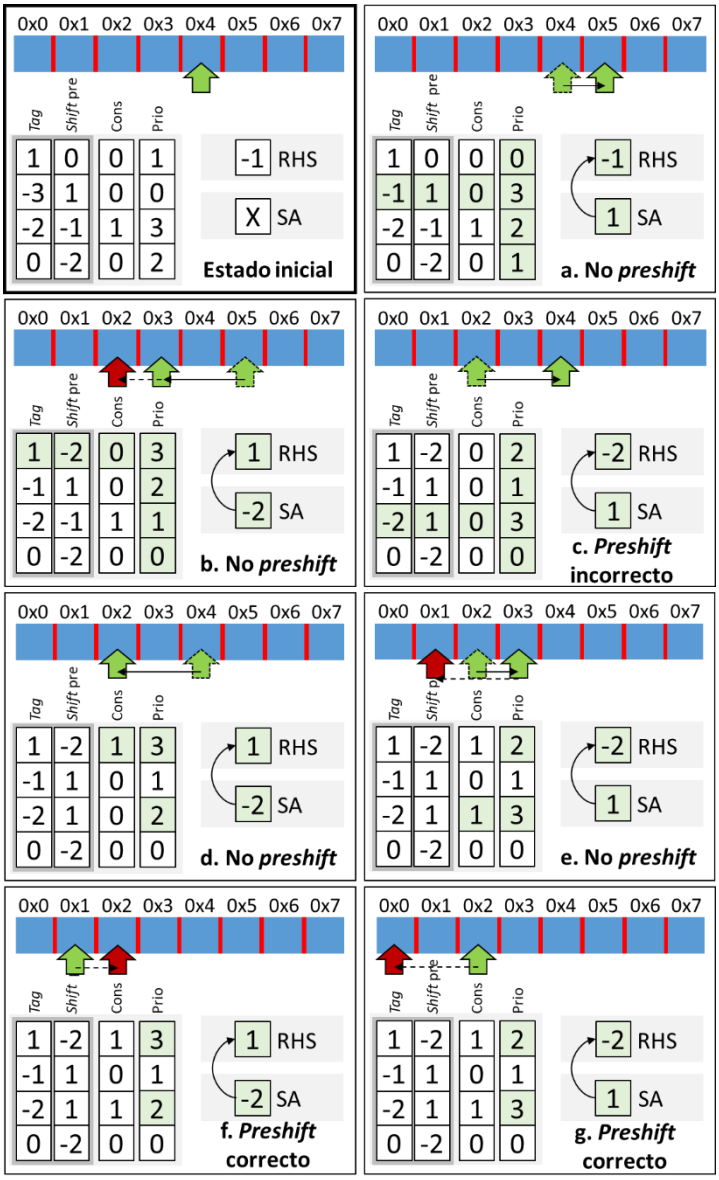

Figura 57 – Ejemplo del funcionamiento del mecanismo propuesto.

Para completar la explicación, se ilustra el funcionamiento del mecanismo mediante un ejemplo que utiliza una configuración simplificada (ver Figura 57). Se emplea un cable con únicamente 8

dominios y un puerto de acceso a los datos, una tabla de patrones con 4 entradas, patrones con una longitud de 2 elementos (compuestos por un *tag* conteniendo un único elemento y el *shift predicho*) y un valor de consolidación de 1. Además, el puerto de acceso se encuentra alineado inicialmente con el dominio cuya dirección es 0x4. El estado inicial de la tabla de patrones, el RHS y el SA está representado en el recuadro superior izquierdo de la Figura 57. A partir del estado de las estructuras, se describe la evolución de su contenido, de la posición del puerto de acceso y de los reposicionamientos realizados para una secuencia de accesos determinada 0x5, 0x3, 0x4, 0x2, 0x3, 0x1, 0x2.

a) El puerto de acceso se encontraba inicialmente alineado con el dominio accedido previamente (0x4). El siguiente acceso hace referencia al dominio 0x5, de manera que se debe realizar una operación de desplazamiento a la derecha (SA = 1) para alinear el puerto con el dominio correspondiente, añadiendo un ciclo extra a la latencia de acceso. En lo que respecta a la actualización de la tabla de patrones, el patrón [-1 1] resultante de la concatenación del valor de RHS (-1) y del SA (1) es insertado en la tabla (RHS -> Tag, SA -> *Shift* predicho). La inserción del nuevo patrón conlleva la expulsión de la entrada con la prioridad más baja ([-3 1] en este caso) y la actualización de los valores de prioridad. Además, el valor del RHS es actualizado con el contenido de SA. En lo concerniente a la predicción del próximo desplazamiento, se encuentra una coincidencia en la tabla (el *tag* de la primera entrada coincide con el valor del RHS). Sin embargo, no se realiza ningún reposicionamiento de la cabeza puesto que el patrón no está consolidado (su valor del campo *consolidación* es 0, mientras que el umbral es 1), de modo que el puerto de acceso se queda alineado con el dominio referencia por el acceso.

b) El siguiente acceso provoca que el puerto de acceso se tenga que desplazar dos posiciones a la izquierda (SA = -2) para alinearse con el dominio 0x3. La búsqueda en la tabla de una entrada cuyo valor del campo *tag* sea idéntico al contenido del RHS tiene éxito, sin embargo, el valor del campo *shift predicho* difiere del contenido de SA. Por lo tanto, el nuevo patrón [1 -2] (RHS + SA) reemplaza al patrón de la tabla con el mismo *tag* que el RHS. El RHS es actualizado apropiadamente al valor -2. En este caso, se encuentra en la tabla un patrón consolidado cuyo valor para el campo *tag* es -2, de modo que se realiza un *preshift* del puerto de acceso de acuerdo con el campo *shift predicho* (1 dominio a la izquierda, flecha roja en la figura).

c) El desplazamiento especulativo del puerto realizado resulta erróneo, dado que el puerto fue desplazado al dominio 0x2, pero el nuevo acceso hace referencia al dominio 0x4, de modo que la predicción errónea añade un ciclo extra a la latencia total. Como consecuencia, el patrón que dio lugar al reposicionamiento especulativo del puerto de acceso es reemplazado con el nuevo patrón ([-2 -1] es reemplazado por [-2 1]). Todo reemplazo implica además el reinicio del valor de consolidación y por supuesto de los valores de prioridad. Con respecto a la predicción, se encuentra una entrada en la tabla cuyo *tag* coincide con el valor de RHS (1), pero no se realiza ningún *preshift* puesto que no está consolidado todavía.

d) El siguiente acceso al dominio 0x2 conlleva la realización de dos operaciones de desplazamiento a la izquierda para alinear debidamente el puerto de acceso y permite la consolidación del patrón [1 -2]. No se realiza ningún *preshift* puesto que el patrón [-2 1] no está consolidado.

e) El puerto de acceso de desplaza una posición a la derecha (0x3) y se consolida el patrón [-2 1]. Además, se reposiciona el puerto de acceso dos posiciones a la izquierda tal y como indica el patrón consolidado [1 -2], alineando el puerto con el dominio 0x1.

f) En este caso, el *preshift* es correcto, de modo que el puerto de acceso está debidamente alineado con el dominio cuando llega al siguiente acceso, eliminando la parte variable de latencia de acceso. Únicamente es necesario actualizar los bits de prioridad. Después de actualizar el contenido del RHS, se realiza un nuevo reposicionamiento del puerto tras encontrar en la tabla de patrones el patrón consolidado [-2 1].

g) El *preshift* resulta ser correcto de nuevo, eliminando igualmente la necesario de alinear el puerto de acceso con el dominio referido por el nuevo acceso. Al igual que en el caso anterior, únicamente hay que actualizar apropiadamente los bits de prioridad. Tras actualizar el contenido del RHS, el puerto se desplaza al dominio 0x0, de acuerdo con el patrón consolidado [1 -2].

5.5.3 Exploración del Espacio de Diseño

Las posibilidades de configuración de la política propuesta motivan la exploración de su sensibilidad a los distintos parámetros de configuración, puesto que, no solo afectan al *shift* promedio, sino que las necesidades de implementación, en términos de área y energía, están directamente relacionadas. En concreto, los principales parámetros a evaluar son: el tipo de patrón, la longitud, el umbral de consolidación y el tamaño de la tabla de patrones. Las aplicaciones que se van a utilizar para la exploración del espacio de diseño son las mismas que las utilizadas en el capítulo anterior, es decir, aplicaciones convencionales (SPEC CPU 2006, PARSEC y NPB) y bases de datos NoSQL (Cassandra, MongoDB, Redis y OrientDB). En las siguientes subsecciones (5.5.3.1 – 5.5.3.4) únicamente se detalla el proceso seguido para determinar la configuración más apropiada en uno de los niveles de la jerarquía *cache* (L2), aunque este proceso se repite en todos los niveles de la jerarquía *cache*, resumiendo la configuración final en la siguiente sección.

5.5.3.1 Tipo de patrón

El funcionamiento del mecanismo de la sección anterior se ha explicado utilizando patrones cuyos elementos eran desplazamientos (o *shifts*). Sin embargo, existen otras alternativas, como la utilización de los identificadores de posición de los dominios para definir los patrones (en una celda con 64 dominios los identificadores van de 0 a 63). Mientras que la primera aproximación se basa en el movimiento relativo del puerto de acceso (distancia y dirección), la segunda utiliza el movimiento absoluto (el identificador numérico del dominio). La utilización de patrones basados en la numeración de los dominios implica que una secuencia de accesos a la *cache* debe hacer referencia a los mismos dominios en al menos dos ocasiones (utilizando umbral de consolidación de 1) para identificar un patrón (la siguiente secuencia de accesos a los dominios 0x4->0x7->0xC->0x4->0x7->0xC se identificaría como el patrón [0x4 0x7 0xC]). Por el contrario, los patrones basados en *shifts* están formados por desplazamientos del puerto de acceso en vez de por los dominios accedidos. Por lo tanto, la identificación de un patrón formado por *shifts* no requiere que estén involucrados los mismos dominios. La secuencia de accesos a la *cache* 0x1->0x2->0x4->0x7->0x8->0xA->0xD (la cual requiere los desplazamientos de la cabeza +1 +2 +3 +1 +2 +3) no se identificaría como un patrón basado en dominios, pero sí como un patrón formado por los siguientes *shifts* [1 2 3])). La elección de un tipo u otro de patrones forma parte de la configuración de la política, de manera que se evalúa el rendimiento de ésta al utilizar ambas opciones para los elementos de los patrones.

La Figura 58 muestra los valores de *shift* promedio obtenidos tras evaluar la política con ambos tipos de patrones para las distintas aplicaciones utilizadas. Los resultados se muestran normalizados a los valores obtenidos al utilizar patrones basados en dominios. En ambas gráficas, las cuatro primeras columnas representan el valor medio del *shift* para las distintas *suites* de *benchmarks* evaluadas (los resultados de SPEC se separan en aplicaciones de enteros y de punto flotante), mientras que la última columna muestra la media geométrica. Los resultados revelan que la política propuesta funciona claramente mejor cuando se utilizan patrones cuyos elementos son *shifts*, un hecho que queda reflejado en la diferencia existente en la última columna de la figura (en particular para las aplicaciones convencionales). La diferencia es especialmente significativa en las aplicaciones pertenecientes a la *suite* NPB (supera el 15%) y en menor medida para SPEC o Cassandra, mientras que prácticamente desaparece (no alcanza el 1%) en las aplicaciones pertenecientes a PARSEC o en el caso de la base de datos OrientDB. Cabe señalar que únicamente hay tres aplicaciones que exhiben mejor rendimiento (*milc* 12% (SPEC), *canneal* 1% (PARSEC) y *facesim* 2,5% (PARSEC)) utilizando patrones de dominios. Se observa que el número de patrones identificados es, en general, mucho más bajo cuando la política funciona con patrones basados en dominios y, que, además, el número de predicciones incorrectas es significativamente mayor. Por el contrario, la utilización de patrones basados en *shifts* permite la identificación de un mayor número de patrones, lo que sugiere que los patrones son válidos para secuencias de accesos que involucran distintos dominios (la distancia entre accesos se mantiene a pesar de acceder a distintas direcciones). En base a estos resultados, se decide la utilización de patrones basados en *shifts* para el resto de los experimentos del presente capítulo de la tesis.

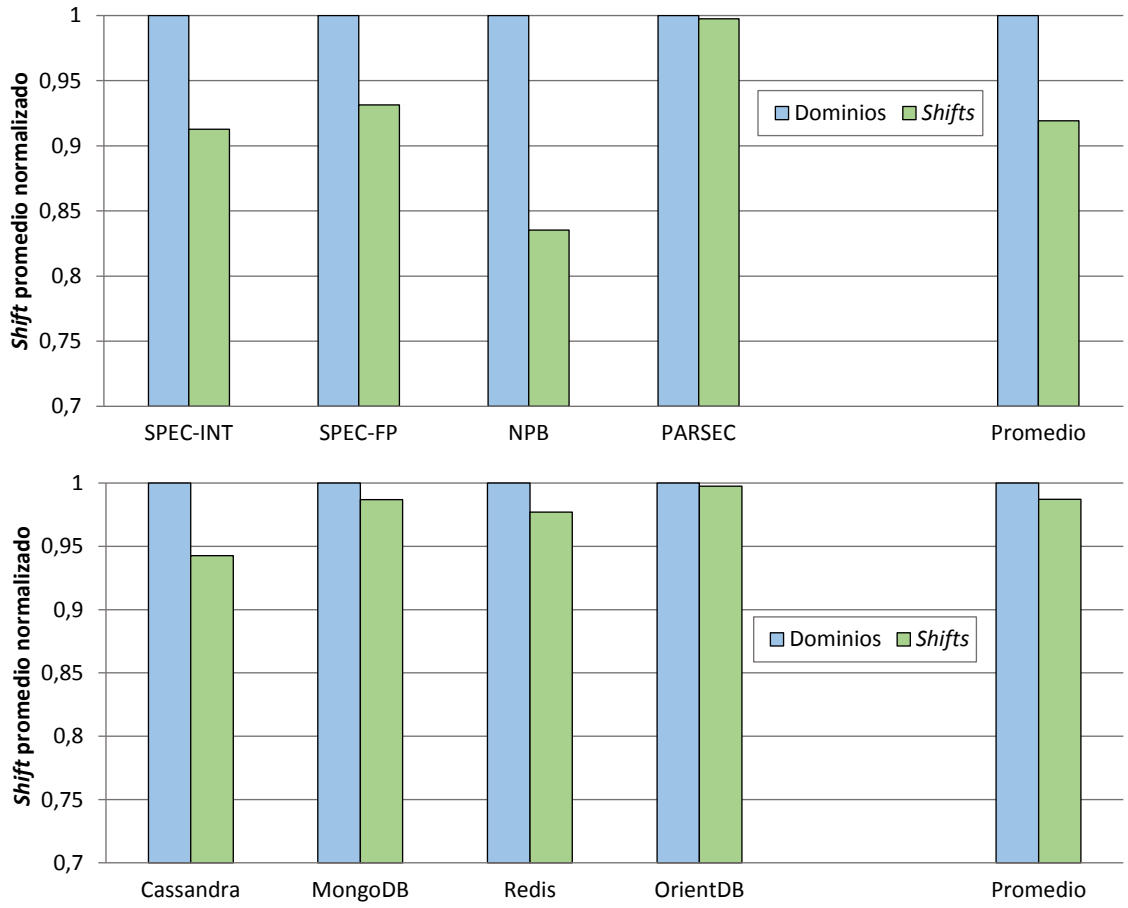

Figura 58 – Shift promedio normalizado comparando los patrones formados por identificadores de dominios y los patrones formados por el desplazamiento del puerto de acceso. (arriba) Aplicaciones convencionales. (abajo) Aplicaciones emergentes.

5.5.3.2 Longitud

La longitud de los patrones es quizás el parámetro más crítico, dado que afecta al número de predicciones, a la precisión de éstas, así como al coste de implementación del mecanismo (la longitud del patrón determina el tamaño de las entradas de la tabla y la energía de las operaciones de actualización de la tabla y de búsqueda del patrón). El uso de un valor bajo para la longitud de los patrones provoca que la precisión disminuya, sin embargo, la utilización de patrones con más elementos hace que se reduzca el número de identificaciones, pues las condiciones para identificar un patrón se endurecen (secuencias más largas de accesos). Por este motivo, es necesario analizar el *trade-off* entre la precisión y la frecuencia de las predicciones para distintos valores de longitud de los patrones. Por otra parte, en lo concerniente a los costes de implementación, la utilización de patrones cortos es más conveniente, puesto que se reduce el área y la energía de la implementación de la tabla de patrones.

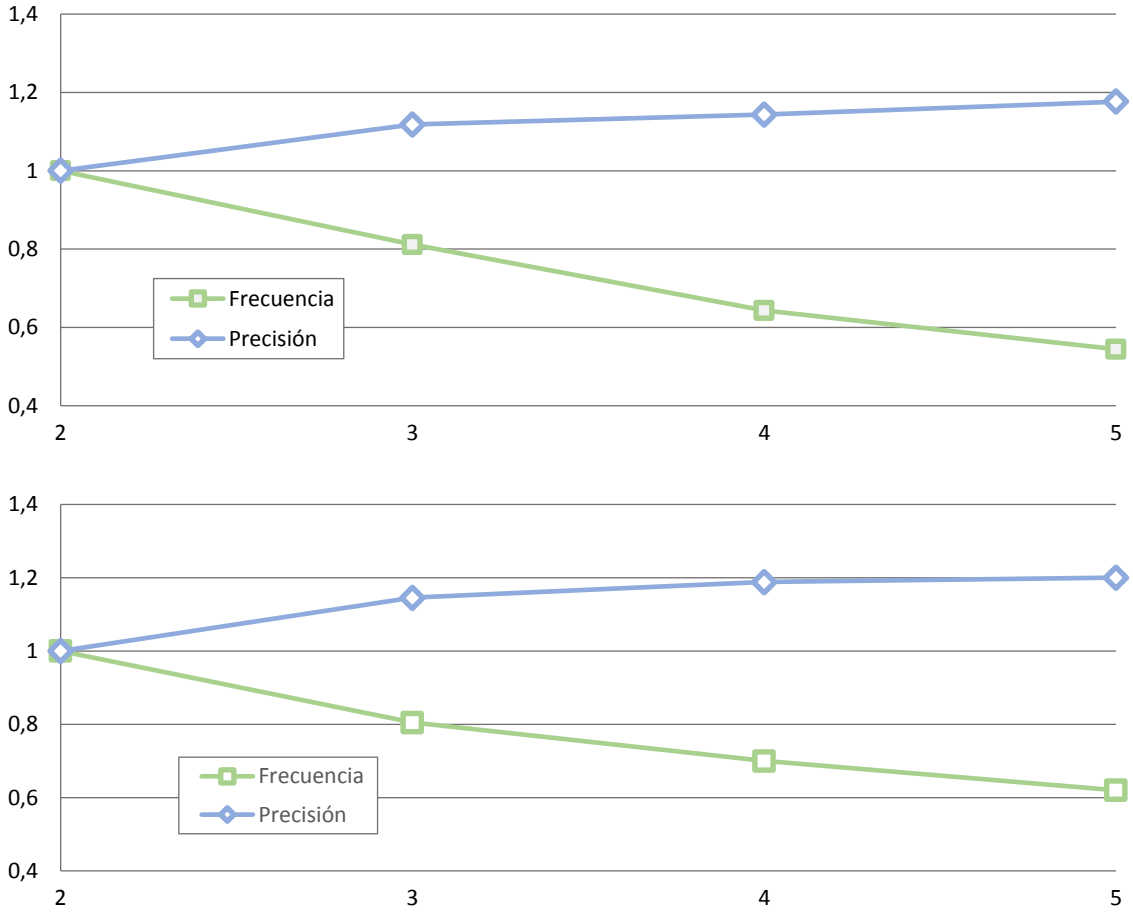

Figura 59 – Frecuencia y precisión de las predicciones normalizadas empleando distintos valores para el parámetro longitud de los patrones. (arriba) Aplicaciones convencionales. (abajo) Aplicaciones emergentes.

Se analiza el rendimiento del mecanismo utilizando cuatro valores distintos para este parámetro, incrementando la longitud de los patrones de 2 a 5. La Figura 59 muestra los resultados obtenidos al evaluar la precisión y la frecuencia de las predicciones, mientras que la Figura 60 muestra su efecto en el *shift* promedio de los accesos. En ambas figuras, los resultados se representan normalizados a los valores obtenidos con el valor de longitud más bajo. Tal y como se esperaba, el uso de patrones con más elementos mejora la precisión de las predicciones, pero al mismo tiempo reduce el número de *preshifts,* un resultado que se repite para los dos tipos de aplicaciones empleados (puesto que la dificultad para que se produzca un acierto en la tabla se endurece a

medida que aumenta el tamaño de los patrones). Los resultados obtenidos con el valor más alto evaluado (5) revelan que el número de predicciones se reduce prácticamente a la mitad en comparación con el valor más bajo (2), mientras que la predicción únicamente mejora un 20%. Este desbalanceo tiene un efecto directo en el rendimiento puesto que la latencia se degrada a medida que disminuye el número de predicciones correctas. El valor promedio de las operaciones de desplazamiento se incrementa con la longitud de los patrones para ambos tipos de aplicaciones, de modo que es preferible la utilización de un valor pequeño para este parámetro. Dado que la longitud de los patrones afecta también al tamaño de la tabla y a la energía de los procesos de actualización y predicción descritos en la sección anterior, se selecciona el valor más pequeño (longitud = 2) para el resto de los experimentos.

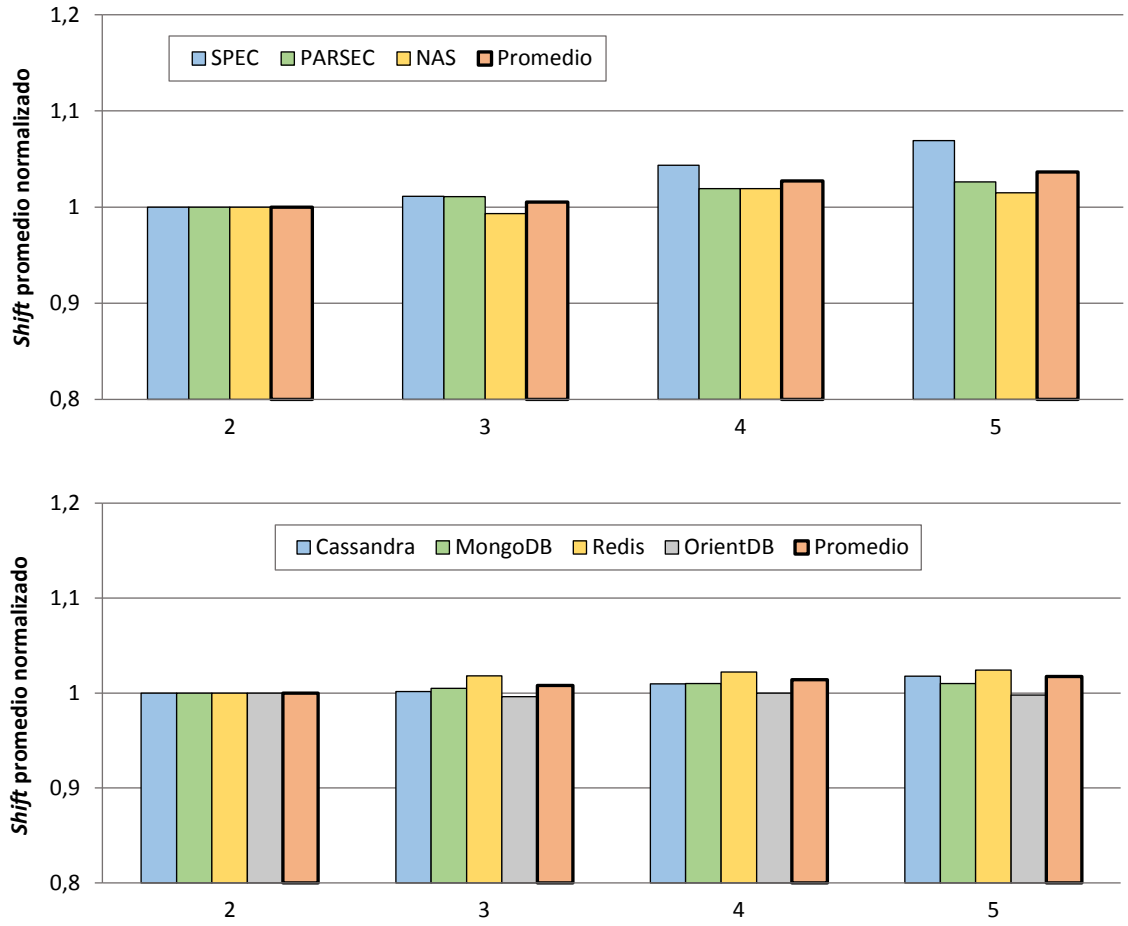

Figura 60 – Shift promedio normalizado al evaluar la política con diferentes valores para el parámetro longitud de los patrones. (arriba) Aplicaciones convencionales. (abajo) Aplicaciones emergentes.

5.5.3.3 Consolidación

El término consolidación hace referencia al número de repeticiones necesarias para considerar la realización de un *preshift* en base a un patrón determinado. De manera similar a la longitud de los patrones, este parámetro tiene influencia directa en la precisión y frecuencia de las predicciones. Un valor grande para la consolidación incrementa la probabilidad de realizar predicciones correctas, pero al mismo tiempo reduce la frecuencia de los reposicionamientos. En concreto, el valor 0 implica que un patrón se considera consolidado tras únicamente una aparición (su inserción en la tabla de patrones). Mientras que, un valor mayor o igual que 1 implica que un patrón debe aparecer al menos dos veces antes de consolidarse. La primera vez que aparece es insertado en la tabla, siendo consolidado tras satisfacer un determinado número de apariciones.

Una vez que el patrón se ha consolidado, la siguiente repetición (parcial) del mismo patrón provoca un *preshift*.

La Figura 61 muestra la sensibilidad de la política a la consolidación, evaluando varios valores para este parámetro, donde los resultados aparecen normalizados a los valores obtenidos con el valor de consolidación más pequeño. En las aplicaciones convencionales, el valor 0 provoca que la política sea demasiado agresiva, degradando la precisión de las predicciones e incrementando en consecuencia la latencia. Por el contrario, los valores mayores que 1 hacen que la política sea demasiado conservadora, lo que provoca a su vez que aumente el *shift* promedio. Esta degradación es muy significativa con respecto al mejor resultado cuando se requieren dos o tres repeticiones para consolidad un patrón, especialmente para las aplicaciones de SPEC y PARSEC. En el caso de las bases de datos NoSQL, el valor 0 para la consolidación también provoca que la política sea demasiado agresiva y a su vez que el *shift* promedio se incremente en comparación con el resto de los valores evaluados. Las diferencias obtenidas en la evaluación de los restantes valores para la consolidación son poco significativas (al contrario que en las aplicaciones convencionales), si bien el valor 1 consigue el *shift* promedio más bajo. Tras los resultados obtenidos, se decide utilizar un valor 1 para la consolidación, de modo que, tras la inserción en la tabla, únicamente se necesita una aparición adicional para consolidar un patrón.

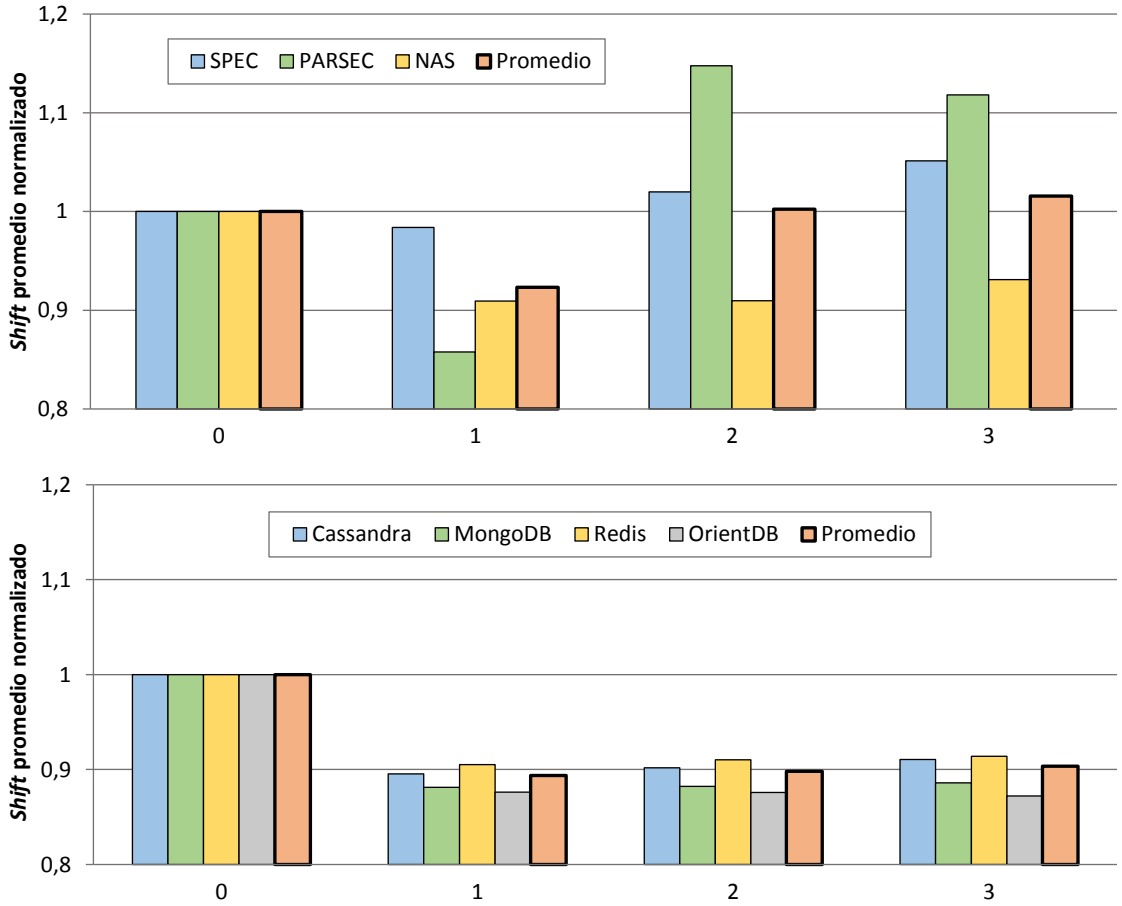

Figura 61 – Shift promedio normalizado al evaluar la política con diferentes valores para el parámetro consolidación. (arriba) Aplicaciones convencionales. (abajo) Aplicaciones emergentes.

5.5.3.4 Capacidad de la tabla

El ultimo parámetro de la configuración a evaluar es el tamaño de la tabla de patrones, es decir, el número de entradas de la tabla. El almacenamiento de todas las posibles configuraciones de los

elementos de los patrones requeriría la implementación de una tabla con un tamaño significativamente grande, incluso utilizando el tamaño más pequeño para la longitud de los patrones (127 filas para patrones con longitud 2), lo que conllevaría un importante *overhead* en términos de área, energía y tiempo de acceso. La implementación del mecanismo que se propone utiliza una tabla con un tamaño limitado, de modo que solo es posible almacenar los N últimos patrones (donde N representa el número de filas de la tabla). En aquellos casos donde es necesario realizar un reemplazo (un patrón de la tabla debe ser expulsado para poder insertar uno nuevo) se utiliza un algoritmo de prioridad basado en la localidad temporal (tipo LRU), de manera similar al reemplazo de bloques de *cache*.

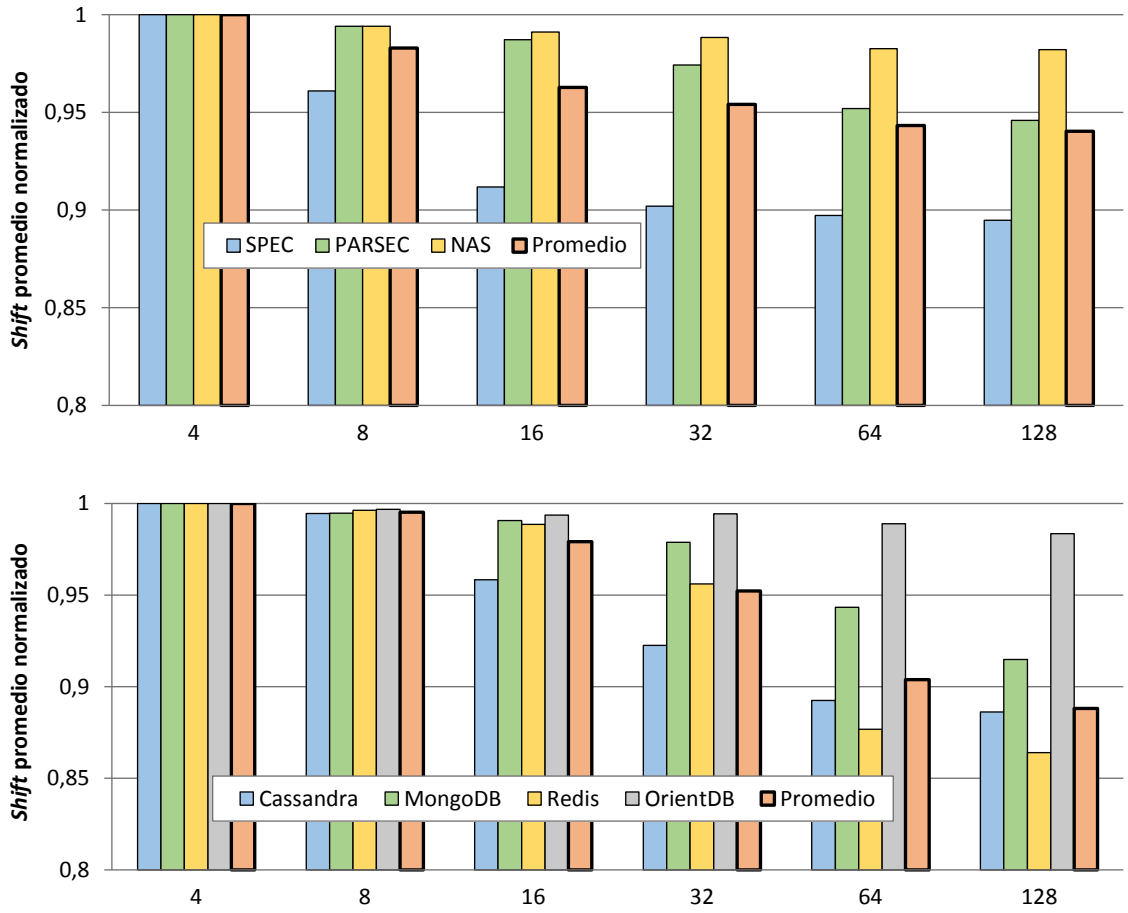

Figura 62 – Shift promedio normalizado al evaluar la política con diferentes valores para el parámetro capacidad. (arriba) Aplicaciones convencionales. (abajo) Aplicaciones emergentes.

Los resultados del experimento para determinar la influencia de la capacidad en el rendimiento están representados en la Figura 62. El análisis se lleva a cabo variando progresivamente la capacidad, desde únicamente 4 entradas en la tabla hasta 128. Los resultados aparecen de nuevo normalizados a los valores obtenidos con el valor más pequeño para el número de entradas en la tabla. Tal y como se esperaba, las distintas aplicaciones experimentan un decrecimiento del *shift* promedio a medida que aumenta la capacidad de la tabla, aunque en diferente grado. En lo referente a las aplicaciones convencionales, la *suite* PARSEC reduce el *shift* promedio un 5% al variar la capacidad de la tabla de 4 a 128 entradas, mientras que en el caso de SPEC la reducción se sitúa por encima del 10%. Sin embargo, el aumento de la capacidad tiene un efecto muy reducido para las aplicaciones NPB, logrando una mejora de únicamente el 2% aproximadamente. En el caso de las aplicaciones NoSQL, los resultados de OrientDB son similares a los de NPB, con

una reducción que no alcanza el 2%. Por el contrario, el resto de las bases de datos sí que logran mejorar sustancialmente el *shift* promedio con el incremento del número de entradas. A la hora de fijar el tamaño de la tabla, es necesario tener en cuenta que la capacidad de ésta es proporcional a su coste de implementación. De este modo, el análisis de los resultados de las aplicaciones convencionales revela que el incremento del tamaño de la tabla de 32 a 128 únicamente produce una mejora en el rendimiento del 1,3%, lo cual no compensa claramente la multiplicación por 4 del número de entradas, ni el coste en área y energía. Por el contrario, la decisión no parece tan clara en el caso de las bases de datos, puesto que, en la mayoría de este conjunto de aplicaciones, el *shift* promedio mejora alrededor de un 10% con el incremento de 32 a 128 entradas en la tabla. Teniendo en cuenta el compromiso entre coste y rendimiento, se decide utilizar un valor de 32, tratando de balancear ambos aspectos. Cabe señalar que, si el coste es abordable, es posible aumentar el rendimiento de la política incrementando el tamaño de la tabla de patrones, en especial para las aplicaciones emergentes.

5.5.4 Configuración final y estimación del coste

El conjunto de experimentos anterior se replica en cada nivel de la *cache* para determinar la configuración más adecuada de la política de reposicionamiento en cada caso, de modo que tras el análisis de los resultados obtenidos se decide utilizar la parametrización del mecanismo que se describe a continuación. La configuración de los parámetros longitud y consolidación son idénticos en todos los niveles, fijando su valor en 2 y 1 respectivamente, de manera que la política trabaja con patrones de *shifts* compuestos por dos elementos y un patrón pasa a estar consolidado tras únicamente dos apariciones (en la primera aparición se inserta en la tabla mientras que en la segunda se consolida). Sin embargo, el tamaño de la tabla varía de un nivel a otro. En concreto, el número de entradas de la tabla para L1 es 16, mientras que para L2 y L3 crece hasta 32 y 64 respectivamente. La configuración del mecanismo de cada nivel está recogida en la Tabla 8.

Tabla 8 – Resumen de la configuración del mecanismo para los distintos niveles.

	Longitud	Consolidación	Capacidad
Dcache	2	1	16
L2	2	1	32
LLC	2	1	64

En lo concerniente al *overhead* en términos de almacenamiento del mecanismo, cada entrada de la tabla necesita la siguiente capacidad para almacenar a información. La codificación de los elementos del patrón requiere 7 bits (complemento a 2) cada uno, puesto que se utilizan celdas con 64 dominios. Además, es necesario un bit adicional para la consolidación y 4 bits para la prioridad en el caso de L1 (en L2 y L3, al aumentar el número de entradas se necesitan 5 y 6 bits por entrada respectivamente para la prioridad). En total, el tamaño de la tabla en L1 es 38 bytes. Dado que cada fila de la *cache* (número de dominios * Tamaño de bloque * Asociatividad) tiene asignada una tabla, el *overhead* en almacenamiento en L1 asciende a 0,1159%. Realizando un cálculo similar en L2 y L3 donde varía el número de bits para la prioridad y la asociatividad se obtiene que el *overhead* es 0,2441% y 0,2563% respectivamente. Para la obtención de la estimación del coste en área y energía, se modela la estructura hardware que implementa la tabla de patrones utilizando DESTINY [200], una herramienta para la exploración del espacio de diseño para SRAM, eDRAM y tecnologías de memoria no volátil similar a CACTI [201] y NVSim [202]. Se obtiene el valor del área, de la energía de las operaciones de lectura/escritura y del *leakage* para la tabla de patrones y el banco de *cache* asociado. A pesar de que la tabla de patrones no se encuentra en el camino crítico del acceso a la *cache*, se asume una implementación orientada al

rendimiento basada en una estructura CAM (*Content-Addressable Memory*), obteniendo valores de peor caso en términos de energía y área. La energía de peor caso en el acceso a la *cache* RM, asumiendo un *shift* promedio de 5 dominios (un valor seleccionado de acuerdo con los resultados observados en la siguiente sección), asciende a 1.5nJ. Por otra parte, la energía consumida en cada acceso (acierto) a la *cache* por parte de la tabla de patrones se sitúa en 0.0088nJ, lo que supone un *overhead* de 0,58% en cada acierto (en los fallos no se accede a la tabla de patrones). Finalmente, en lo concerniente al área, la implementación de la tabla de patrones supone un *overhead* del 2,89%. La estimación de área y energía de un banco de *cache* y de la tabla de patrones y energía está recogida en la Tabla 9.

Tabla 9 – Estimación de área y energía.

	Banco de *cache* 32KB	Tabla de patrones
Area (mm^2)	0,2007	0,0058
Energía lectura (nJ)	0,3506	0,0056
Energía escritura (nJ)	0,1491	0,0032
Energía *shift* (nJ)	0,2310	-
Leakage (mW)	207	7,12

5.5.5 Evaluación de Rendimiento

En esta sección se evalúa el rendimiento de la propuesta frente a las políticas de referencia (LAZY [51], EAGER [51] y NEXT-BLOCK [52]) en los distintos niveles de la jerarquía de memoria. La evaluación de las aplicaciones tradicionales (SPEC, PARSEC y NPB (sección 3.2.1)) y emergentes (YCSB (sección 3.3.1)) se lleva a cabo simulando uno y dos CMPs con 4 *cores* con la configuración de la Tabla 10, siguiendo la metodología propuesta anteriormente, basada en la utilización de *checkpoints* y en la simulación de una porción de la ROI de las aplicaciones. En lo que respecta a la tecnología RM, se utilizan celdas con 64 dominios y un único puerto de acceso. Además, el desplazamiento del puerto a un dominio adyacente conlleva 1 ciclo de penalización.

Tabla 10 – Configuración del core y de la jerarquía cache.

Arquitectura del *core*	Unidades funcionales	4xI-ALU / 4xFP-ALU/ 4xD-Mem
	Tamaño del ROB / Anchura del *issue*	128 / 4
	Frecuencia/ Número	3Ghz / 4
Caches privadas	(L1) Tamaño / Asociatividad / Tamaño de bloque / Tiempo de acceso	32KB I / 32KB D (128KB RM) / 4-vías / 64B / 2 ciclos
	(L2) Tamaño / Asociatividad / Tamaño de bloque / Tiempo de acceso	256KB (4MB RM) Unificada (I+D) / 8-vías / 64B / 10 ciclos / Exclusiva con L1
LLC	Tamaño / Asociatividad / Tamaño de bloque / Tiempo de acceso	16 MB (64MB RM) / 16 vías / 64B / 24 ciclos / Mostly inclusive
	Protocolo de coherencia / Modelo de consistencia / *Prefetch*	MOESI *snooping* / TSO / *Tagged prefetcher*
RM	Dominios / Puertos de acceso / Velocidad de desplazamiento	64 / 1 RW / 1 ciclo/dominio
Memory	Capacidad / Tiempo de acceso / Ancho de banda	4GB / 250 ciclos / 32GB/s

5.5.5.1 Shift promedio

El primer conjunto de resultados de la evaluación de rendimiento analiza el *shift* promedio asociado al conjunto de políticas de reposicionamiento en los distintos niveles de la jerarquía de memoria *on-chip*. Los resultados de la evaluación de las aplicaciones tradicionales se muestran en la Figura 63, donde aparecen normalizados a los valores de LAZY y las aplicaciones están ordenadas de mejor a peor rendimiento para facilitar la interpretación de los resultados. Los resultados obtenidos revelan un comportamiento diferente de las políticas de referencia en cada uno de los niveles. En la Dcache, las políticas EAGER y NEXT-BLOCK son incapaces de mejorar el

rendimiento de LAZY (únicamente EAGER logra mejorar el rendimiento de LAZY en una aplicación), debido a que en este nivel de la *cache* hay un alto grado de localidad en el acceso a los datos, lo que claramente beneficia a la política LAZY. Por el contrario, la política que se propone sí es capaz de batir a LAZY, aunque sea de manera moderada. La mejora que obtiene la política propuesta, denominada "PATTERN", se sitúa por encima del 5% para únicamente 5 *workloads*, reduciendo el *shift* promedio hasta un 12% en el mejor de los casos y siendo ligeramente peor que LAZY para 6 *workloads.*

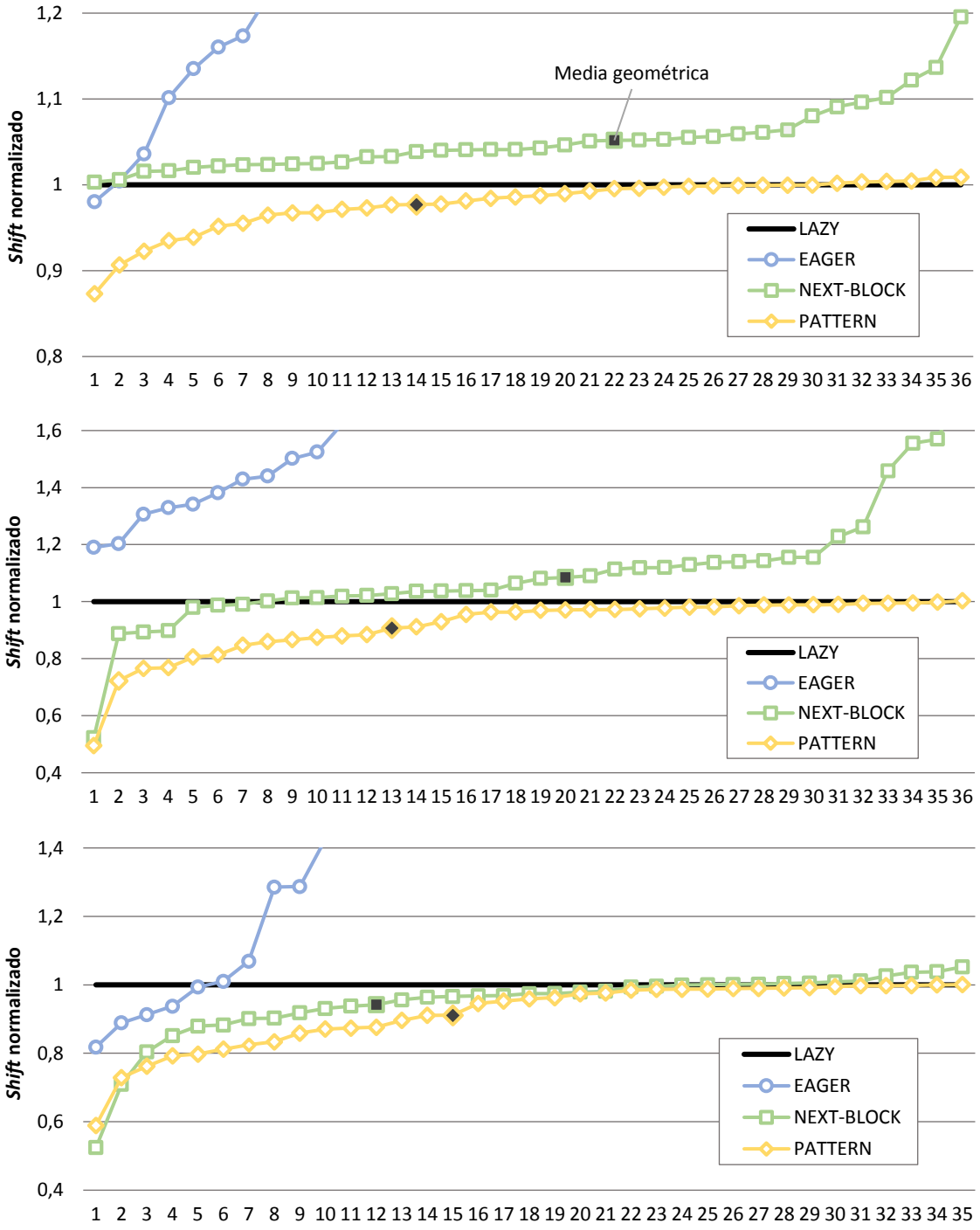

Figura 63 – Shift promedio de las políticas evaluadas (resultados normalizados a los valores de LAZY) para las aplicaciones tradicionales. Una gráfica por cada nivel de cache: (arriba) Dcache. (medio) L2. (abajo) LLC. Los workloads están ordenados de mejor a peor rendimiento.

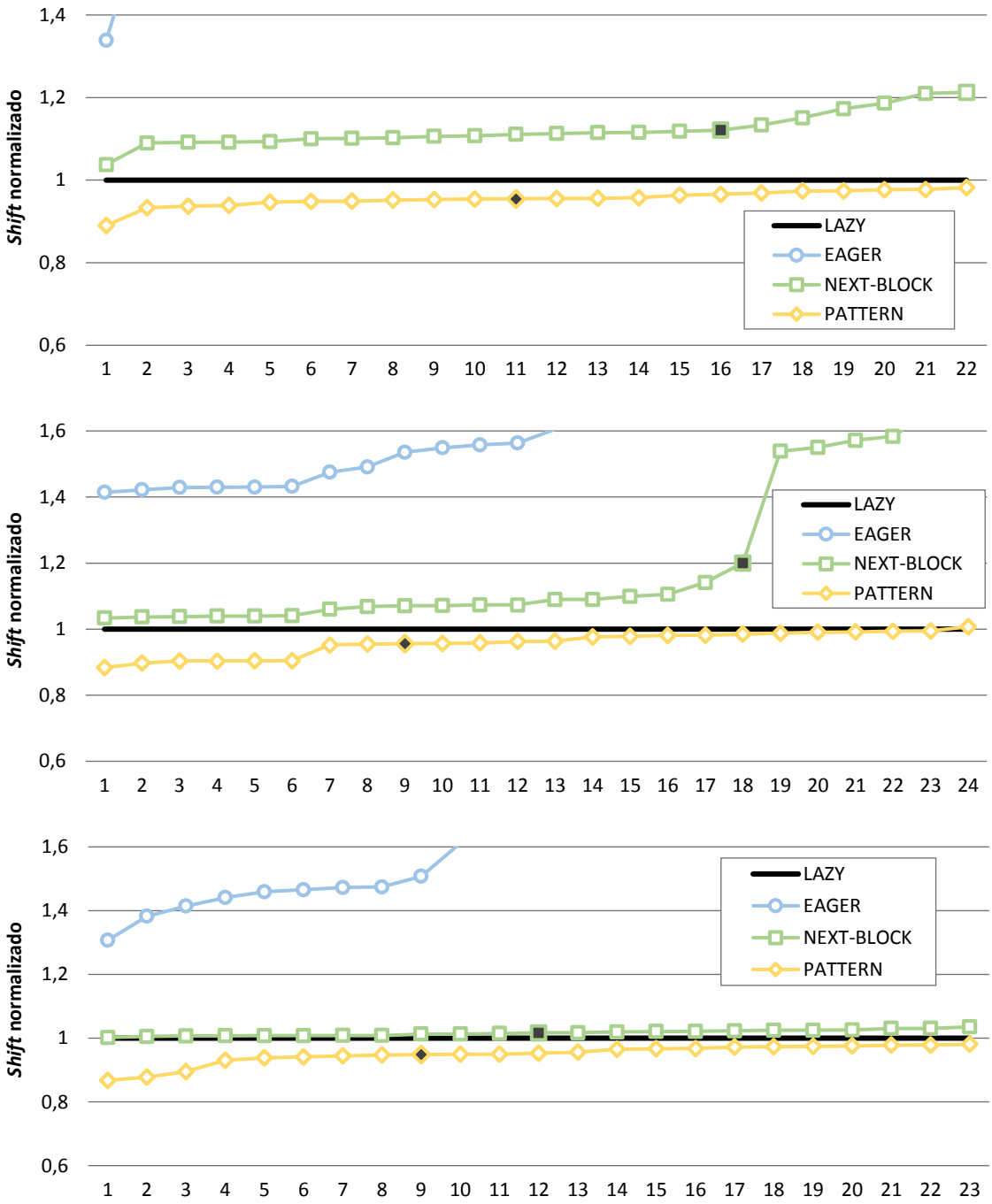

Figura 64 – Shift promedio de las políticas evaluadas (resultados normalizados a los valores de LAZY) para las aplicaciones emergentes. Una gráfica por cada nivel de cache: (arriba) Dcache. (medio) L2. (abajo) LLC. Los workloads están ordenados de mejor a peor rendimiento.

Los resultados en L2 y en LLC, donde parte de la localidad ha sido filtrada, reducen el rendimiento de la política LAZY. Además, ambos elementos de almacenamiento son compartidos por datos e instrucciones y, en el caso de la LLC, es compartido a su vez por todos los *cores* del chip. Por estos motivos, la variabilidad de los accesos a la *cache* se incrementa, lo que perjudica a las políticas estáticas (LAZY y NEXT-BLOCK). Al comparar LAZY y NEXT-BLOCK se puede apreciar que el *shift* promedio depende de cada aplicación y de su patrón de acceso, siendo difícil determinar cuál de las dos políticas es mejor. En L2, LAZY obtiene mejor rendimiento que NEXT-BLOCK (NEXT-BLOCK únicamente es capaz de batir a LAZY en 7 aplicaciones), sin embargo, la situación en LLC es la

contraria, siendo NEXT-BLOCK mejor que LAZY. No obstante, PATTERN es capaz de batir de nuevo a ambas políticas en los dos niveles de *cache*. En L2, mejora el rendimiento de LAZY al ejecutar la mitad de los *workloads* en un 5% aproximadamente y alcanza una reducción del *shift* promedio del 50% en el mejor de los casos. En LLC, los resultados son similares, con la diferencia de que la significativa mejora de NEXT-BLOCK sobre LAZY le acerca a los resultados de PATTERN.

Los resultados obtenidos al evaluar las distintas políticas con las bases de datos NoSQL se muestran en la Figura 64, donde de nuevo los resultados se representan normalizados a los valores obtenidos con LAZY y ordenados de mejor a peor rendimiento. Estos resultados revelan un comportamiento distinto al de las aplicaciones tradicionales. De nuevo, LAZY se alza como la mejor política de referencia, ya que, en este caso, EAGER y NEXT-BLOCK se muestran incapaces de batir su rendimiento en los tres niveles evaluados. Las diferencias entre estas tres políticas varían en función del nivel donde de evalúen, pero los resultados son consistentes. Cabe destacar, que al igual que sucede con las aplicaciones tradicionales, el rendimiento de NEXT-BLOCK mejora en LLC en comparación con los restantes niveles, pero en este caso no logra batir a LAZY. Por el contrario, PATTERN es capaz de batir consistentemente a las tres políticas, si bien la ganancia con respecto a LAZY decrece en comparación con las aplicaciones tradicionales. En la Dcache, PATTERN mejora en más de un 5% a LAZY para 7 aplicaciones, reduciendo la latencia de desplazamiento en un 11% en el mejor de los casos. Además, su rendimiento no empeora con respecto a LAZY en ninguna aplicación. En L2, PATTERN es capaz de mejorar el rendimiento de LAZY en un 10% para 6 aplicaciones (correspondientes a los *workloads* A-E en la base de datos Cassandra), y, en este caso, se empeora el rendimiento de LAZY en una ocasión (WC en OrientDB). Por último, en LLC, PATTERN mejora a LAZY en más de un 5% al ejecutar los distintos *workloads* sobre Redis y OrientDB y no empeora el rendimiento el LAZY en ningún caso. Tras la obtención de los resultados descritos, se puede concluir que la política de reposicionamiento basada en la identificación de patrones de acceso es capaz de mejorar consistentemente el rendimiento de las políticas propuestas en la literatura con independencia del nivel de *cache* analizado. El rendimiento depende de la existencia de patrones en el acceso a los datos por parte de las aplicaciones, siendo marginales los casos donde el rendimiento de la política propuesta es peor que el de la mejor de las políticas de la literatura y, además, en esos casos la degradación es despreciable.

Los resultados de los experimentos anteriores han demostrado que la política propuesta es capaz de batir de manera consistente al resto de políticas para el conjunto de aplicaciones evaluado al utilizar un número alto de dominios (64 concretamente), una decisión fundamentada en aras de aprovechar al máximo la densidad de integración de la tecnología RM. Sin embargo, es posible la utilización de esta tecnología con un menor número de bits por celda, siendo necesario determinar cómo influye la variación del número de dominios en la identificación de patrones por parte de la política. Por este motivo, se extiende la evaluación del *shift* promedio al utilizar valores alternativos para el número de bits por celda, limitando los experimentos a las aplicaciones convencionales para no extender en exceso esta parte de la evaluación. La Figura 65 muestra el *shift* promedio obtenido al evaluar para las políticas LAZY y PATTERN en el segundo nivel de la jerarquía utilizando 16, 32 y 64 dominios (los valores correspondientes a 64 dominios son los del experimento anterior), mostrándose los resultados normalizados a los valores obtenidos con LAZY. Tal y como es posible apreciar, PATTERN consigue batir a LAZY con independencia del número de dominios evaluado. De hecho, la mejora de PATTERN sobre LAZY se acentúa a medida que disminuye el número de dominios puesto que, al reducirse el rango de los posibles dominios, el mecanismo es capaz de detectar más patrones y, en consecuencia, reposicionar apropiadamente el puerto de acceso. Cabe señalar que el incremento en el porcentaje de mejora con respecto a LAZY al disminuir el número de dominios supone un menor ahorro en ciclos con

respecto a los resultados anteriores, debido a que la magnitud del *shift* promedio disminuye de acuerdo con el número de dominios en cada celda.

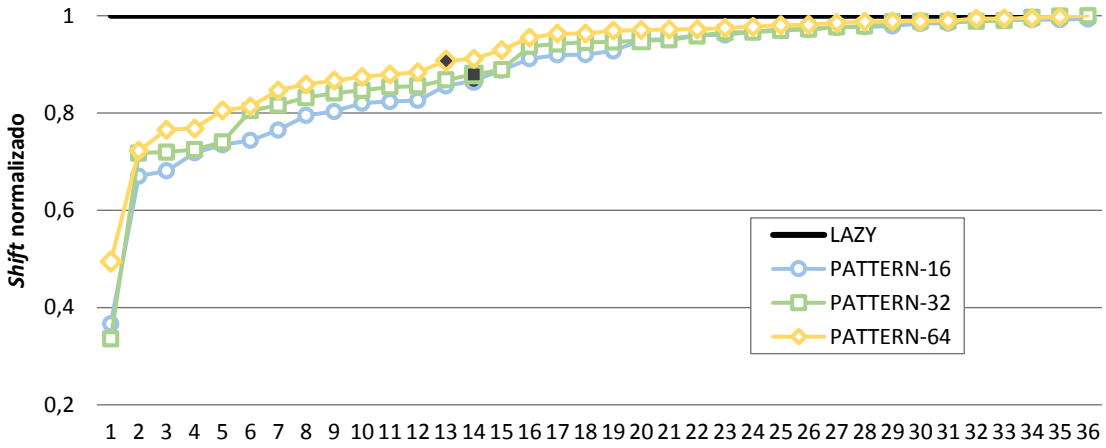

Figura 65 – Shift promedio en L2 para las políticas LAZY y PATTERN utilizando distintos valores para el número de dominios y las aplicaciones convencionales. Los workloads están ordenados de mejor a peor rendimiento.

Aunque no se muestran los resultados obtenidos al repetir este experimento en los otros dos niveles de la jerarquía (Dcache y LLC), los resultados obtenidos en el segundo nivel de la *cache* son extensibles al resto de niveles. De igual manera, el *shift* promedio de PATTERN con respecto a LAZY disminuye con el número de dominios tanto en la Dcache como en la LLC. El conjunto de experimentos llevado a cabo en esta sección del capítulo sugiere que PATTERN es consistentemente mejor que el resto de políticas presentes en la literatura con independencia del nivel de la jerarquía de memoria donde se aplique y del número de dominios utilizado (no se ha evaluado el rendimiento de las distintas políticas con un número de dominios inferior a 16 pues se considera que la tecnología RM va ligada a un número de dominios suficiente que compense las dificultades de la implementación de esta tecnología).

5.5.5.2 Memory latency

Los experimentos de la subsección anterior han demostrado que la política propuesta es capaz de reducir la latencia promedio de las operaciones de desplazamiento en todos los niveles de la jerarquía de memoria *on-chip* para el conjunto de aplicaciones evaluado. Gracias al reposicionamiento del puerto que permite el acceso a los bits antes de la llegada siguiente acceso, la parte variable de la latencia de acceso puede reducirse significativamente en muchos casos. En esta parte de la evaluación se va un a paso más allá para determinar el impacto que esta reducción tiene en la latencia promedio de acceso a memoria. Se descarta la utilización de RM en el primer nivel de *cache* puesto que la latencia de acceso es crítica, de modo que el *overhead* de las operaciones de desplazamiento perjudica notablemente el rendimiento. Además, dado que la latencia de L1 tiene un valor fijo, su inclusión en la métrica no aporta información relevante para el rendimiento de las diferentes políticas evaluadas. Por este motivo, se escoge el acceso a memoria promedio observado para acceder a los niveles implementados con RM (equivalente al tiempo promedio de fallo de L1) como métrica de rendimiento para las políticas evaluadas, incluyendo la latencia de acceso a memoria principal.

La Figura 66 muestra los resultados (normalizados de nuevo a los valores de LAZY) del acceso promedio a los elementos de la jerarquía de memoria implementados con la tecnología RM para el mismo conjunto de políticas y de aplicaciones de los experimentos anteriores. La comparación de las políticas propuestas en la literatura arroja el mismo resultado de los experimentos anteriores, donde LAZY se destaca como la mejor política propuesta en la literatura (motivo por

el que forma parte de la política propuesta). EAGER y NEXT-BLOCK se muestran incapaces de mejorar la latencia de LAZY para la mayoría de las aplicaciones evaluadas. A pesar de que el rendimiento de NEXT-BLOCK en la LLC es similar al de LAZY (ver Figura 66) para las aplicaciones convencionales, su mal funcionamiento en L2 degrada la latencia promedio de acceso. Sin embargo, el buen rendimiento de la política propuesta en los distintos niveles lleva a superar el rendimiento de LAZY. Para el conjunto de aplicaciones analizado, PATTERN obtiene mejor rendimiento en 58 de las 59 aplicaciones evaluadas. La mejora supera el 5% en 10 aplicaciones, situándose el caso más favorable en el 10%. No obstante, se observa una reducción en el margen de mejora entre este experimento y los anteriores. Cabe señalar que la mejora en la latencia alcanzable depende en gran medida de la existencia de patrones en el acceso a los datos en L2 y en LLC, así como de la tasa de fallos en LLC (un alto número de accesos a memoria principal puede ocultar los beneficios de la política de reposicionamiento). Este hecho implica que, para maximizar el beneficio de la política, ambos niveles deben mostrar patrones en el acceso a los datos y los fallos en LLC deberían ser mínimos. En cualquier caso, para el conjunto de aplicaciones analizado, la política es consistentemente mejor que LAZY, siendo destacable el hecho de que prácticamente no se empeora en ningún caso.

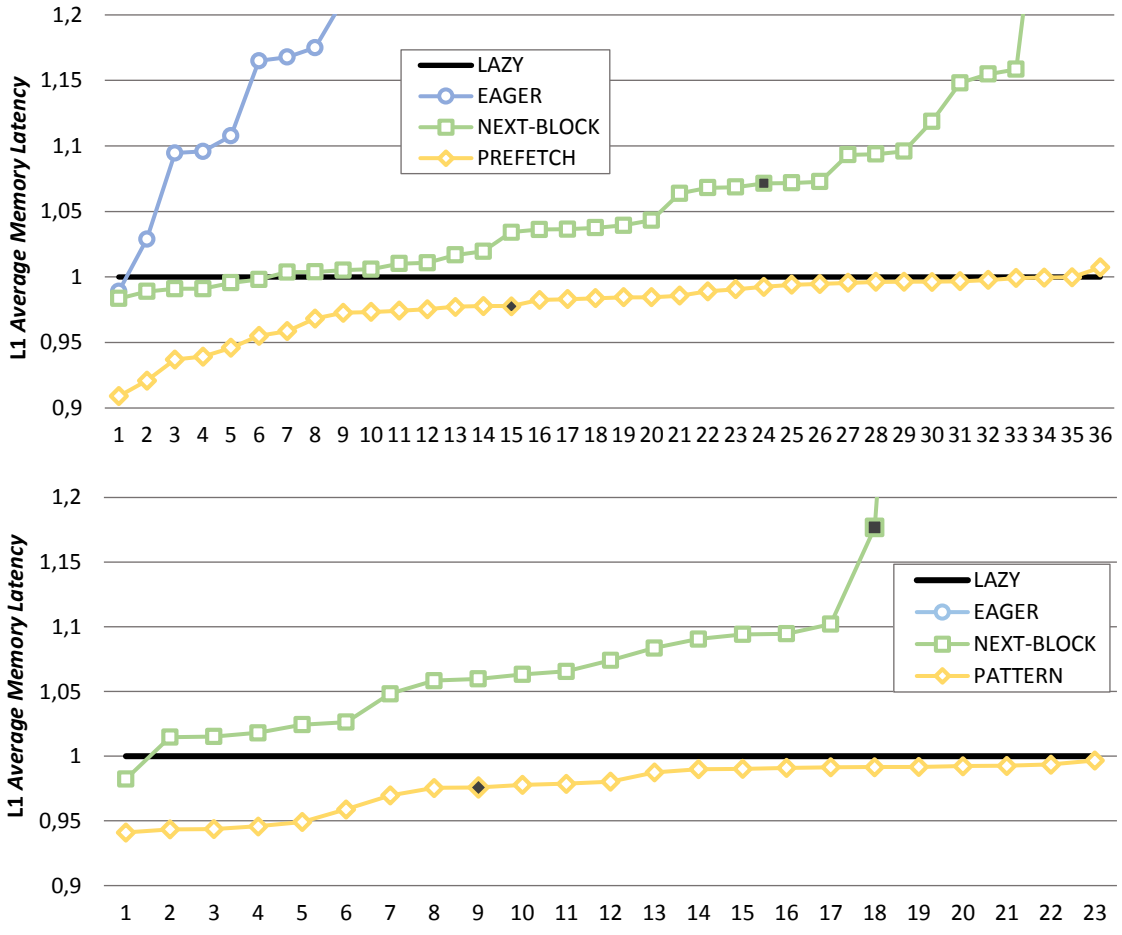

Figura 66 – Latencia promedio de acceso a memoria de L1. (arriba) Aplicaciones convencionales. (abajo) Aplicaciones emergentes.

5.5.6 RM como reemplazo a tecnologías convencionales (SRAM)

Para terminar este capítulo de la tesis, se lleva a cabo una comparación a fin de determinar el rendimiento de la tecnología RM frente a SRAM. La comparación de estas dos tecnologías en los trabajos publicados hace uso de herramientas de simulación tecnológicas para obtener valores de

área, energía y tiempos de acceso. Sin embargo, estas herramientas parecen ser poco fiables, un hecho fundamentado en los distintos valores que se pueden observar en los trabajos ya publicados. Tal discrepancia se manifiesta en los trabajos [51][192], donde la utilización de distintas herramientas (CACTI y NVSim respectivamente) para la simulación de los parámetros tecnológicos arroja distintos resultados para la misma capacidad y nodo tecnológico. Ante esta incertidumbre, se decide hacer una comparación de SRAM y RM de manera agnóstica a la tecnología. En este escenario, se supone que la tecnología RM logra equiparar los tiempos de acceso de SRAM, y se busca determinar si el posible incremento de la densidad de integración (bits/mm^2) que ofrece la RM es capaz de paliar la naturaleza variable de la latencia de los accesos. Cabe señalar, que los beneficios de la tecnología RM, en términos área y energía, frente a SRAM están fuera de toda duda. El menor número de transistores empleados para la construcción de las celdas y el mayor número de bits por celda permiten una amplia densidad de integración. Además, la significativa reducción del *leakage* debido al descenso del número de transistores empleado es mayor que el *overhead* en la energía dinámica de los accesos y de las operaciones de desplazamiento [51][188][190][192].

Este último experimento comienza evaluando el rendimiento de SRAM y RM en igualdad de condiciones en términos capacidad (y latencia), para, a continuación, incrementar paulatinamente la capacidad de la RM (asumiendo una densidad de almacenamiento creciente). Al igual que en los experimentos de la sección anterior, la tecnología RM se aplica en los niveles segundo y tercero (descartando su utilización en el primer nivel de la jerarquía), y se utiliza un número de dominios que asciende a 64.

La Figura 67 muestra la evolución del IPC promedio de las distintas aplicaciones evaluadas al incrementar la capacidad de la *cache on-chip*. Los resultados se representan normalizados a los valores obtenidos para SRAM y la configuración tomada como base (L2: 256Kb – L3: 8MB). Así, los resultados etiquetados como “RMx1” corresponden a las simulaciones donde la capacidad es la misma que la empleada en la evaluación de la SRAM, los resultados “RMx2” doblan la capacidad de SRAM (L2: 512kb – L3: 16MB), etc. Adicionalmente, las barras de error muestran el IPC máximo y mínimo (no se incluyen para las aplicaciones emergentes puesto que no aportan información debido a la homogeneidad de los resultados de los distintos *workloads*). Los resultados de las aplicaciones convencionales muestran que el incremento de densidad de integración que debe ofrecer la tecnología RM con respecto a SRAM para paliar el *overhead* en los tiempos de acceso se situaría entre 4 (SPEC y NPB) y 8 (PARSEC) veces mayor capacidad para un área equivalente. A partir de dicho nivel, la creciente densidad de integración compensaría el *overhead* de latencia de manera significativa. Además, las barras de error revelan que el IPC mínimo se sitúa más cerca del valor promedio de SRAM que el IPC máximo, lo que indica que en determinadas aplicaciones la tecnología RM logra superar con un alto margen el rendimiento de SRAM. Estos resultados parecen prometedores, dadas las estimaciones del potencial incremento en densidad de integración de la tecnología RM [160]. En el caso de las aplicaciones emergentes, la RM necesitaría mayor densidad de almacenamiento para superar el rendimiento de SRAM al ser evaluada con las bases de datos NOSQL. En este caso, la multiplicación de la capacidad de SRAM por 16 no logra igualar los valores de rendimiento, siendo necesario ir más allá. Estas aplicaciones presentan un *shift* promedio más alto que el de las aplicaciones convencionales, de modo que, es más difícil compensar el *overhead* en la latencia con la capacidad.

Los resultados de este experimento sugieren que, si la tecnología RM es capaz de ofrecer unos tiempos de lectura/escritura semejantes a los de SRAM, proporcionando un incremento de la densidad de almacenamiento de aproximadamente un orden de magnitud, puede ser competitiva

en términos de rendimiento en comparación con las actuales, todo ello con las ventajas ya mencionadas sobre consumos energéticos.

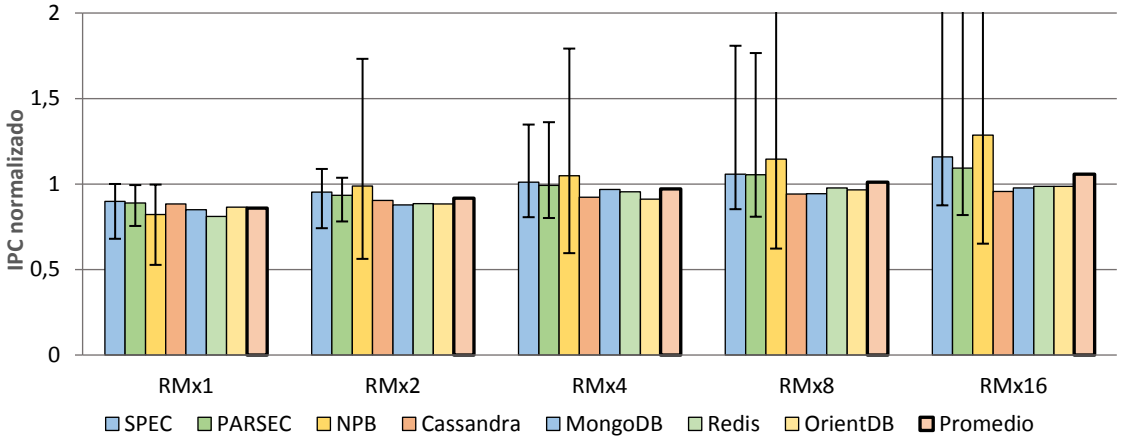

Figura 67 – Evolución del IPC (normalizado) de la tecnología RM a medida que se incrementa la capacidad.

5.6 Conclusiones

En este capítulo, se ha realizado la propuesta de una solución micro-arquitectural para acercar la integración de una prometedora tecnología de memoria no volátil dentro de la jerarquía de *cache on-chip*. Las bondades de la tecnología RM con respecto a SRAM o DRAM, como son la mayor densidad de integración o menor *leakage*, tienen como contrapartida la existencia de determinados desafíos, como la naturaleza variable de los accesos. El trabajo realizado en la parte final de esta tesis se ha centrado precisamente en este aspecto, proponiendo y evaluando una política de reposicionamiento de la cabeza lectora entre accesos. Esta política utiliza un mecanismo basado en una técnica de *prefetch hardware* para la identificación de patrones en el acceso a los datos. De este modo, ante la identificación de un patrón, es posible el apropiado reposicionamiento del puerto de acceso antes de la llegada de la siguiente referencia a memoria, consiguiendo reducir o eliminar el *overhead* asociado a la necesidad de alinear el puerto con el dominio que se desea acceder.

Tras analizar con detalle la sensibilidad de la política a los parámetros de configuración, se ha evaluado su rendimiento, tomando como referencia políticas alternativas propuestas en la literatura. Los resultados han demostrado (para el amplio conjunto de aplicaciones evaluado) que la política propuesta es capaz de batir consistentemente a las políticas de referencia, con independencia del nivel de *cache* donde se evalúe (aunque en distinto grado), consiguiendo reducir satisfactoriamente el *shift* promedio de los accesos en la mayoría de las aplicaciones evaluadas, ya sean convencionales o emergentes. En determinadas aplicaciones, esta reducción alcanza el 50% y, cabe destacar que, la política propuesta no empeora prácticamente en ningún caso el mejor rendimiento observado en las políticas de referencia. Adicionalmente, se ha explorado la influencia del número de dominios en el rendimiento de la política, obteniendo unos resultados igualmente positivos, donde el beneficio de la propuesta no solo se mantiene, sino que crece ligeramente a medida que disminuye el número de dominios. La reducción del *shift* promedio se traduce, a su vez, en una reducción del tiempo promedio de acceso que perciben las *caches* de primer nivel al utilizar la tecnología RM en los niveles segundo y tercero de la jerarquía de *cache*.

Finalmente, la comparación de las tecnologías SRAM y RM, en un escenario donde los tiempos de acceso de la tecnología RM han logrado igualar a los de SRAM, revela que el aumento de la capacidad que permite la tecnología RM logra paliar el *overhead* que introduce la variabilidad de la latencia de los accesos, consiguiendo mejorar el rendimiento que ofrece la tecnología SRAM. El punto en el que la capacidad logra compensar el *overhead* varía con la aplicación, obteniendo unos resultados que indican que las aplicaciones tradicionales sacan mayor provecho de la capacidad disponible que las bases de datos NoSQL.

6 Conclusiones y Trabajo Futuro

6.1 Conclusiones

En esta sección, se describen brevemente las principales conclusiones extraídas de este trabajo.

6.1.1 *Benchmarking* de aplicaciones emergentes

El progresivo crecimiento de la cuota de mercado de las aplicaciones emergentes motiva su incorporación a la evaluación en el área de Arquitectura de Computadores. Los requerimientos que plantean estas aplicaciones a la metodología más utilizada en el área, la simulación, no son despreciables, pues se ha demostrado la importancia del sistema operativo o de la correcta simulación de estas aplicaciones (entorno distribuido). Tales requerimientos elevan la complejidad de las herramientas, y, en consecuencia, incrementan el coste computacional de las simulaciones y los recursos hardware (memoria) asociados. Al mismo tiempo, la correcta preparación de las cargas de trabajo también se vuelve una tarea más compleja, tal y como se ha visto en este trabajo. En definitiva, la incorporación de las nuevas aplicaciones supone un desafío para la metodología utilizada y una ardua tarea para los investigadores.

6.1.2 Caracterización de aplicaciones emergentes

El trabajo de caracterización realizado ha permitido lograr un mayor entendimiento, con respecto a otros trabajos de caracterización que utilizan metodologías con claras limitaciones, de las necesidades de las bases de datos NoSQL en la jerarquía de memoria. Los resultados que se han obtenido sugieren que, en determinados aspectos, el comportamiento de las bases de datos y de las aplicaciones convencionales no difiere en exceso. Así, la presencia de localidad en los accesos provoca que ambos tipos de aplicaciones se beneficien del aumento de la capacidad. Sin embargo, estas aplicaciones han mostrado ciertas peculiaridades que podrían ser exploradas en aras de desarrollar jerarquías más acordes a sus necesidades particulares. Por otra parte, resulta sorprendente la significativa homogeneidad de los resultados, ya sea entre los distintos *workloads* para cada base de datos o de manera global entre el conjunto de bases de datos, un hecho que se debe potencialmente al complejo *stack* software compartido por las cuatro bases de datos evaluadas.

6.1.3 Tecnologías de memoria no volátil

La creciente presión sobre la jerarquía de memoria *on-chip*, debido al mayor número de *cores* presentes en el chip y a los cada vez mayores *working sets* de las aplicaciones, unido a los límites de la escalabilidad CMOS, ha motivado el interés sobre vías alternativas para seguir escalando el rendimiento de esta parte del hardware. Así, en los últimos años se ha hecho un gran esfuerzo en el desarrollo de tecnologías de memoria no volátil, un conjunto de tecnologías con notables características, pero no exento de desafíos. El trabajo realizado con una tecnología emergente en la parte final de esta tesis ha demostrado cómo la Arquitectura de Computadores puede mitigar los desafíos existentes en estas tecnologías emergentes. En este caso particular, se ha realizado una propuesta para minimizar la naturaleza variable de los accesos en la tecnología RM, mediante el uso de una política de reposicionamiento del puerto de acceso a los datos que emplea un mecanismo similar al utilizado en el *prefetch* hardware para la identificación de patrones en los accesos. La extensa evaluación con aplicaciones convencionales y emergentes sugiere que la propuesta realizada es capaz de identificar patrones en el acceso a los datos para reposicionar

adecuadamente el puerto de acceso, siendo consistentemente mejor que las políticas existentes en la literatura, con independencia del nivel de la jerarquía *on-chip* donde se aplique y del número de bits presente en las celdas. Sin duda, al margen de las mejoras tecnológicas, la Arquitectura de Computadores tendrá un papel relevante en la integración de estas tecnologías en el chip.

6.2 Trabajo Futuro

En esta sección, se describen las principales líneas de investigación futuras, relacionadas con el trabajo de caracterización y la tecnología de memoria no volátil llevado a cabo.

6.2.1 Caracterización de aplicaciones emergentes

La especialización software que ha tenido lugar en los últimos años no se ha limitado únicamente al desarrollo de bases de datos NoSQL como las que se han caracterizado en este trabajo, sino que software alternativo para el almacenamiento y procesado de datos como Hadoop o Spark, múltiples servicios web (búsqueda, redes sociales, *streaming*) o aplicaciones de *Machine Learning* también forman parte del conjunto de aplicaciones emergentes. Asimismo, los resultados obtenidos en la caracterización de las bases de datos NoSQL llevada a cabo en este trabajo no tienen por qué extenderse al resto de aplicaciones, de modo que se antoja necesario continuar caracterizando el software novedoso, para así poder diseñar arquitecturas acordes a sus requerimientos y lograr maximizar el rendimiento y minimizar el consumo.

Por otra parte, en el trabajo de caracterización realizado, el número de nodos simulados se ha limitado a dos, separando únicamente el cliente de la parte del servidor. Sin embargo, algunas de las aplicaciones que se han caracterizado, como es el caso de Mongodb, tienen arquitecturas donde la parte del servidor se descompone en múltiples nodos con diferentes roles. Este hecho motiva la continuación del trabajo de caracterización para así determinar si la interacción de los distintos roles en la parte del servidor añade cierto margen de error en las medidas, tal y como sucede con la interacción entre el cliente y el servidor demostrada en este trabajo.

6.2.2 Consolidación dinámica

La exploración del espacio de diseño de la política propuesta se ha realizado analizando los múltiples resultados de las simulaciones en toda su extensión. Tales resultados no significan necesariamente que la configuración finalmente seleccionada sea la más adecuada en todo momento, puesto que es posible que en determinadas fases de las aplicaciones sea más beneficiosa la utilización de valores alternativos para los parámetros de configuración. De este modo, la utilización de la política con carácter variable podría llevar a un mejor funcionamiento de la misma y, en consecuencia, a una mayor reducción del *shift* promedio que la obtenida en este trabajo, mejorando el rendimiento. En esta línea, el parámetro más proclive a ser modificado durante la ejecución es la consolidación, puesto que con únicamente dos bits por cada entrada de la tabla permitiría la configuración de este parámetro en un rango de 0 a 3, haciendo que la política funcionase de un modo más agresivo o conservador según las necesidades variables de la aplicación.

7 Bibliografía

[1] E. Seeram, *Computed Tomography-E-Book: Physical Principles, Clinical Applications, and Quality Control*. Elsevier Health Sciences, 2015.

[2] M. J. Mack, "Minimally invasive and robotic surgery," *Jama*, vol. 285, no. 5, pp. 568–572, 2001.

[3] T. A. C. CENTER, "CANCER RESEARCH: A SUPERCOMPUTING PERSPECTIVE." [Online]. Available: https://www.tacc.utexas.edu/special-report/cancer. [Accessed: 05-Oct-2018].

[4] M. S. K. C. Center, "Watson Oncology." [Online]. Available: https://www.mskcc.org/about/innovative-collaborations/watson-oncology. [Accessed: 04-Oct-2018].

[5] G. E. Moore, "Cramming More Components Onto Integrated Circuits," *Proc. IEEE*, vol. 86, no. 1, pp. 82–85, Jan. 1998.

[6] "5 nanometer transistors inching their way into chips." [Online]. Available: https://www.ibm.com/blogs/think/2017/06/5-nanometer-transistors/. [Accessed: 21-Oct-2018].

[7] M. D. Hill *et al.*, "21st century computer architecture," *arXiv Prepr. arXiv1609.06756*, 2016.

[8] W. Aspray, "The Intel 4004 microprocessor: what constituted invention?," *IEEE Ann. Hist. Comput.*, vol. 19, no. 3, pp. 4–15, 1997.

[9] I. Cutress, "Intel Launches 7th Generation Kaby Lake: 15W/28W with Iris, 35-91W Desktop and Mobile Xeon." [Online]. Available: https://www.anandtech.com/show/10959/intel-launches-7th-generation-kaby-lake-i7-7700k-i5-7600k-i3-7350k. [Accessed: 04-Oct-2018].

[10] INTERNATIONAL ROADMAP FOR DEVICES AND SYSTEMS (IRDS), "MORE MOORE," 2017.

[11] M. M. Waldrop, "The chips are down for Moore's law," *Nature*, vol. 530, no. 7589, pp. 144–147, Feb. 2016.

[12] D. Hisamoto *et al.*, "FinFET-a self-aligned double-gate MOSFET scalable to 20 nm," *IEEE Trans. Electron Devices*, vol. 47, no. 12, pp. 2320–2325, 2000.

[13] "Samsung Starts Mass Production of its 2nd Generation 10nm FinFET Process Technology," 2017. [Online]. Available: https://news.samsung.com/global/samsung-starts-mass-production-of-its-2nd-generation-10nm-finfet-process-technology. [Accessed: 20-Sep-2018].

[14] M. Lapedus, "Transistor Options Beyond 3nm." [Online]. Available: https://semiengineering.com/transistor-options-beyond-3nm/. [Accessed: 20-Sep-2018].

[15] S. E. Thompson and S. Parthasarathy, "Moore's law: the future of Si microelectronics," *Mater. Today*, vol. 9, no. 6, pp. 20–25, Jun. 2006.

[16] J. Wills, "7 Ways Amazon Uses Big Data to Stalk You." [Online]. Available: https://www.investopedia.com/articles/insights/090716/7-ways-amazon-uses-big-data-stalk-you-amzn.asp. [Accessed: 22-Oct-2018].

[17] D. Borthakur, "The hadoop distributed file system: Architecture and design," *Hadoop Proj. Website*, vol. 11, no. 2007, p. 21, 2007.

[18] Oracle, “Sun QFS File System 5.3 Configuration and Administration Guide.” [Online]. Available: https://docs.oracle.com/cd/E22586_01/html/E22571/toc.html. [Accessed: 21-Oct-2018].

[19] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” *Commun. ACM*, vol. 51, no. 1, p. 107, Jan. 2008.

[20] “Apache Spark.” [Online]. Available: https://spark.apache.org/. [Accessed: 04-Oct-2018].

[21] N. Leavitt, “Will NoSQL Databases Live Up to Their Promise?,” *Computer (Long. Beach. Calif).*, vol. 43, no. 2, pp. 12–14, Feb. 2010.

[22] A. Lakshman and P. Malik, “Cassandra: a decentralized structured storage system,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, p. 35, Apr. 2010.

[23] “MongoDB.” [Online]. Available: https://www.mongodb.com/es. [Accessed: 04-Oct-2018].

[24] M. Abadi *et al.*, “Tensorflow: a system for large-scale machine learning.,” in *OSDI*, 2016, vol. 16, pp. 265–283.

[25] “PyTorch.” [Online]. Available: https://pytorch.org/. [Accessed: 21-Sep-2018].

[26] N. P. Jouppi *et al.*, “In-Datacenter Performance Analysis of a Tensor Processing Unit,” in *Proceedings of the 44th Annual International Symposium on Computer Architecture - ISCA ’17*, 2017, pp. 1–12.

[27] M. Ferdman *et al.*, “Clearing the clouds,” in *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS ’12*, 2012, p. 37.

[28] L. Wang *et al.*, “BigDataBench: A big data benchmark suite from internet services,” in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 488–499.

[29] A. Colaso *et al.*, “Memory Hierarchy Characterization of NoSQL Applications through Full-System Simulation,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 5, pp. 1161–1173, May 2018.

[30] R. Panda, C. Erb, M. LeBeane, J. H. Ryoo, and L. K. John, “Performance Characterization of Modern Databases on Out-of-Order CPUs,” in *2015 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2015, pp. 114–121.

[31] A. J. Awan, M. Brorsson, V. Vlassov, and E. Ayguade, “Performance Characterization of In-Memory Data Analytics on a Modern Cloud Server,” in *2015 IEEE Fifth International Conference on Big Data and Cloud Computing*, 2015, pp. 1–8.

[32] M. Malik, S. Rafatirah, A. Sasan, and H. Homayoun, “System and architecture level characterization of big data applications on big and little core server architectures,” in *2015 IEEE International Conference on Big Data (Big Data)*, 2015, pp. 85–94.

[33] B. Black *et al.*, “Die Stacking (3D) Microarchitecture,” in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO’06)*, 2006, pp. 469–479.

[34] J. D. Owens *et al.*, “A Survey of General-Purpose Computation on Graphics Hardware,” *Comput. Graph. Forum*, vol. 26, no. 1, pp. 80–113, Mar. 2007.

[35] P. Ye, “Switching channels,” *IEEE Spectr.*, vol. 53, no. 12, pp. 40–45, Dec. 2016.

[36] J. A. del Alamo, “Nanometre-scale electronics with III–V compound semiconductors,”

Nature, vol. 479, no. 7373, pp. 317–323, Nov. 2011.

[37] N. Singh *et al.*, “High-performance fully depleted silicon nanowire (diameter /spl les/ 5 nm) gate-all-around CMOS devices,” *IEEE Electron Device Lett.*, vol. 27, no. 5, pp. 383–386, May 2006.

[38] M. Motoyoshi, “Through-Silicon Via (TSV),” *Proc. IEEE*, vol. 97, no. 1, pp. 43–48, Jan. 2009.

[39] K. N. Tu, “Reliability challenges in 3D IC packaging technology,” *Microelectron. Reliab.*, vol. 51, no. 3, pp. 517–523, Mar. 2011.

[40] “Samsung Starts Mass Producing Industry’s First 128-Gigabyte DDR4 Modules for Enterprise Servers,” 2015. [Online]. Available: https://news.samsung.com/global/samsung-starts-mass-producing-industrys-first-128-gigabyte-ddr4-modules-for-enterprise-servers. [Accessed: 04-Oct-2018].

[41] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, “Dark Silicon and the End of Multicore Scaling,” *IEEE Micro*, vol. 32, no. 3, pp. 122–134, May 2012.

[42] A. Chen, “A review of emerging non-volatile memory (NVM) technologies and applications,” *Solid. State. Electron.*, vol. 125, pp. 25–38, Nov. 2016.

[43] M. Kund *et al.*, “Conductive bridging RAM (CBRAM): an emerging non-volatile memory technology scalable to sub 20nm,” in *IEEE InternationalElectron Devices Meeting, 2005. IEDM Technical Digest.*, pp. 754–757.

[44] M. Hosomi *et al.*, “A novel nonvolatile memory with spin torque transfer magnetization switching: spin-ram,” in *IEEE InternationalElectron Devices Meeting, 2005. IEDM Technical Digest.*, pp. 459–462.

[45] B. Tallis, “IBM And Everspin Announce 19TB NVMe SSD With MRAM Write Cache.” [Online]. Available: https://www.anandtech.com/show/13174/ibm-and-everspin-announce-19tb-nvme-ssd-with-mram-write-cache. [Accessed: 20-Sep-2018].

[46] Everspin, “Everspin Begins 40nm STT-MRAM Volume Production,” 2018. [Online]. Available: https://www.everspin.com/sites/default/files/pressdocs/Everspin_Begins_STT-MRAM_Volume_Ramp-011718.pdf. [Accessed: 21-Oct-2018].

[47] D. Kanter, “Adesto Targets IoT Using CBRAM,” 2016.

[48] S. S. P. Parkin, M. Hayashi, and L. Thomas, “Magnetic Domain-Wall Racetrack Memory,” *Science (80-.).*, vol. 320, no. 5873, pp. 190–194, Apr. 2008.

[49] S. Yu and P.-Y. Chen, “Emerging Memory Technologies: Recent Trends and Prospects,” *IEEE Solid-State Circuits Mag.*, vol. 8, no. 2, pp. 43–56, 2016.

[50] A. Colaso, P. Prieto, P. Abad, V. Puente, and J. A. Gregorio, “Architecting RM preshift through pattern-based prediction mechanisms,” *Submitt. (Revision progress)*.

[51] R. Venkatesan, V. Kozhikkottu, C. Augustine, A. Raychowdhury, K. Roy, and A. Raghunathan, “TapeCache: a high density, energy efficient cache based on domain wall memory,” in *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design - ISLPED ’12*, 2012, p. 185.

[52] R. Venkatesan, M. Sharad, K. Roy, and A. Raghunathan, “DWM-TAPESTRI - An Energy Efficient All-Spin Cache Using Domain Wall Shift Based Writes,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, 2013, pp. 1825–1830.

[53] SPEC Standard Performance Evaluation Corporation, “SPEC 2006.” [Online]. Available: https://spec.org. [Accessed: 28-Sep-2018].

[54] C. Bienia, S. Kumar, J. P. Singh, and K. Li, “The PARSEC benchmark suite,” in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques - PACT ’08*, 2008, p. 72.

[55] L. Eeckhout, *Computer Architecture Performance Evaluation Methods*. 2010.

[56] N. Binkert *et al.*, “The Gem5 Simulator,” *{SIGARCH} Comput. Arch. News*, vol. 39, no. 2, pp. 1–7, 2011.

[57] A. Colaso, P. Prieto, J. Á. Herrero, P. Abad, V. Puente, and J.-Á. Gregorio, “Accuracy vs. Computational Cost Tradeoff in Distributed Computer System Simulation,” *Submitt. (Revision progress)*.

[58] International Data Corporation (IDC), “THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadow s, and Biggest Grow th in the Far East,” 2012.

[59] IBM, “Global Technology Outlook 2012.”

[60] International Telecommunication Union, “Percentage of Individuals using the Internet.” [Online]. Available: www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx. [Accessed: 23-Oct-2018].

[61] “Google now handles at least 2 trillion searches per year.” [Online]. Available: https://searchengineland.com/google-now-handles-2-999-trillion-searches-per-year-250247. [Accessed: 21-Sep-2018].

[62] “YouTube chief claims ‘best year ever,’ plays down Facebook threat.” [Online]. Available: https://www.cio.com/article/2952482/video/youtube-chief-claims-best-year-ever-plays-down-facebook-threat.html. [Accessed: 21-Sep-2018].

[63] “Facebook reveals we upload a whopping 350 million photos to the network daily.” [Online]. Available: https://www.digitaltrends.com/social-media/according-to-facebook-there-are-350-million-photos-uploaded-on-the-social-network-daily-and-thats-just-crazy/. [Accessed: 21-Sep-2018].

[64] A. Nordrum, “Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated.” [Online]. Available: https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated. [Accessed: 23-Oct-2018].

[65] Ericsson, “Ericsson Mobility Report,” 2015.

[66] D. Laney, “3D Data Management: Controlling Data Volume, Velocity, and Variety,” 2001.

[67] Y. Demchenko, P. Grosso, C. de Laat, and P. Membrey, “Addressing big data issues in Scientific Data Infrastructure,” in *2013 International Conference on Collaboration Technologies and Systems (CTS)*, 2013, pp. 48–55.

[68] J. Ginsberg, M. H. Mohebbi, R. S. Patel, L. Brammer, M. S. Smolinski, and L. Brilliant, “Detecting influenza epidemics using search engine query data,” *Nature*, vol. 457, no. 7232, pp. 1012–1014, Feb. 2009.

[69] B. Marr, “Big Data: How Netflix Uses It to Drive Business Success.” [Online]. Available: https://www.smartdatacollective.com/big-data-how-netflix-uses-it-drive-business-success/. [Accessed: 22-Oct-2018].

[70] “Big Data Delivers Big Results at UPS.” [Online]. Available: https://pressroom.ups.com/pressroom/ContentDetailsViewer.page?ConceptType=Speeches&id=1426415450350-355. [Accessed: 22-Oct-2018].

[71] C. L. Philip Chen and C.-Y. Zhang, “Data-intensive applications, challenges, techniques and technologies: A survey on Big Data,” *Inf. Sci. (Ny).*, vol. 275, pp. 314–347, Aug. 2014.

[72] N. Zanoon, A. Al-Haj, and S. M. Khwaldeh, “Cloud Computing and Big Data is there a Relation between the Two: A Study,” *Int. J. Appl. Eng. Res.*, vol. 12, no. 17, pp. 6970–6982, 2017.

[73] T. White, *Hadoop: The Definitive Guide*. 2009.

[74] D. Kim, “The Hadoop Vs. the NoSQL - Whiteboard Walkthrough.” [Online]. Available: https://mapr.com/blog/hadoop-vs-nosql-whiteboard-walkthrough/. [Accessed: 20-Sep-2018].

[75] Dhruba Borthakur, “HDFS Architecture Guide,” *Apache Softw. Found.*, 2008.

[76] J. Sen Sarma, “Hadoop,” 2008. [Online]. Available: https://code.facebook.com/posts/423120391138341/hadoop/. [Accessed: 21-Sep-2018].

[77] M. Lopez, “How eBay Uses Big Data and Machine Learning to Drive Business Value,” 2017. [Online]. Available: https://lopezresearch.com/how-ebay-uses-big-data-and-machine-learning-to-drive-business-value/. [Accessed: 21-Sep-2018].

[78] “How LinkedIn uses Hadoop to leverage Big Data Analytics?” [Online]. Available: https://www.dezyre.com/article/how-linkedin-uses-hadoop-to-leverage-big-data-analytics/229. [Accessed: 21-Sep-2018].

[79] N. Hemsoth, “THE YAHOO BEHIND FRESH DEEP LEARNING APPROACHES AT FLICKR,” 2015. [Online]. Available: https://www.nextplatform.com/2015/09/03/the-yahoo-behind-fresh-deep-learning-approaches-at-flickr/. [Accessed: 21-Sep-2018].

[80] “Limitations of Hadoop, Ways to Resolve Hadoop Drawbacks.” [Online]. Available: https://techvidvan.com/tutorials/limitations-of-hadoop-and-solutions/. [Accessed: 21-Sep-2018].

[81] “DB-Engines Ranking.” [Online]. Available: https://db-engines.com/en/ranking. [Accessed: 05-Jun-2018].

[82] “NoSQL Market is Expected to Reach $4.2 Billion, Globally, by 2020 - Allied Market Research.” [Online]. Available: https://www.prnewswire.com/news-releases/nosql-market-is-expected-to-reach-42-billion-globally-by-2020---allied-market-research-498476751.html. [Accessed: 21-Sep-2018].

[83] S. P. E. Corporation, “SPEC Cloud® IaaS 2016.” [Online]. Available: https://www.spec.org/cloud_iaas2016/. [Accessed: 21-Sep-2018].

[84] MarkLogic, “Why an Enterprise NoSQL Database for Unstructured Information,” 2012.

[85] V. Abramova, J. Bernardino, and P. Furtado, “Experimental evaluation of NoSQL databases,” *Int. J. Database Manag. Syst.*, vol. 6, no. 3, p. 1, 2014.

[86] F. Chang *et al.*, “Bigtable: A distributed storage system for structured data,” *ACM Trans. Comput. Syst.*, vol. 26, no. 2, p. 4, 2008.

[87] G. DeCandia *et al.*, “Dynamo: amazon’s highly available key-value store,” in *ACM SIGOPS*

operating systems review, 2007, vol. 41, no. 6, pp. 205–220.

[88] "Key-Value Database Explained." [Online]. Available: http://basho.com/resources/key-value-databases/. [Accessed: 21-Sep-2018].

[89] "Redis." [Online]. Available: https://redis.io/. [Accessed: 20-Sep-2018].

[90] "Voldemort." [Online]. Available: http://www.project-voldemort.com/voldemort/. [Accessed: 20-Sep-2018].

[91] "CouchDB." [Online]. Available: http://couchdb.apache.org/. [Accessed: 20-Sep-2018].

[92] "OrientDB." [Online]. Available: https://orientdb.com/. [Accessed: 20-Sep-2018].

[93] "neo4j." [Online]. Available: https://neo4j.com/. [Accessed: 20-Sep-2018].

[94] "ArangoDB." [Online]. Available: https://www.arangodb.com/. [Accessed: 20-Sep-2018].

[95] L. George, *HBase: the definitive guide: random access to your planet-size data*. " O'Reilly Media, Inc.," 2011.

[96] L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M. Lou Soffa, "The impact of memory subsystem resource sharing on datacenter applications," in *Proceeding of the 38th annual international symposium on Computer architecture - ISCA '11*, 2011, p. 283.

[97] Intel, "Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3," 2016.

[98] A. C. De Melo, "The new linux'perf'tools," *Slides from Linux Kongress*, vol. 18, 2010.

[99] "Intel® VTune™ Amplifier." [Online]. Available: https://software.intel.com/en-us/intel-vtune-amplifier-xe. [Accessed: 22-Oct-2018].

[100] A. Yasin, "A Top-Down method for performance analysis and counters architecture," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014, pp. 35–44.

[101] D. Chiou *et al.*, "FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle-Accurate Simulators," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, 2007, pp. 249–261.

[102] R. A. Uhlig and T. N. Mudge, "Trace-driven memory simulation: a survey," *ACM Comput. Surv.*, vol. 29, no. 2, pp. 128–170, Jun. 1997.

[103] D. R. Kaeli, "Issues in Trace-Driven Simulation," in *Performance Evaluation of Computer and Communication Systems*, 1993, pp. 224–244.

[104] A. Gutierrez *et al.*, "Sources of error in full-system simulation," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014, pp. 13–22.

[105] A. Butko, R. Garibotti, L. Ost, and G. Sassatelli, "Accuracy evaluation of GEM5 simulator system," in *7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, 2012, pp. 1–7.

[106] B. M. A. S. Bischoff, A. Sandberg, A. Hansson, D. Sunwoo, A.G. Saidi, M. Horsnell, "Flexible and High-Speed System-Level Performance Analysis using Hardware-Accelerated Simulation," *Des. Autom. Test Eur.*, 2013.

[107] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Tenth international conference on architectural support for programming languages and operating systems on Proceedings of the 10th international*

conference on architectural support for programming languages and operating systems (ASPLOS-X) - ASPLOS '02, 2002, p. 45.

[108] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1, Aug. 2011.

[109] M. M. K. Martin *et al.*, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4. p. 92, 2005.

[110] P. S. Magnusson *et al.*, "Simics: A full system simulation platform," *Computer (Long. Beach. Calif).*, vol. 35, no. 2, pp. 50–58, 2002.

[111] K. C. Yeager, "The Mips R10000 superscalar microprocessor," *IEEE Micro*, vol. 16, no. 2, pp. 28–41, Apr. 1996.

[112] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11*, 2011, p. 1.

[113] J. E. Miller *et al.*, "Graphite: A distributed parallel simulator for multicores," in *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, 2010, pp. 1–12.

[114] C.-K. Luk *et al.*, "Pin: building customized program analysis tools with dynamic instrumentation," in *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation - PLDI '05*, 2005, p. 190.

[115] D. Genbrugge, S. Eyerman, and L. Eeckhout, "Interval simulation: Raising the level of abstraction in architectural simulation," in *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, 2010, pp. 1–12.

[116] P. Ortego and P. Sack, "SESC: SuperESCalar Simulator," 2004.

[117] J. E. Veenstra and R. J. Fowler, "MINT: a front end for efficient simulation of shared-memory multiprocessors," in *Proceedings of International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 201–207.

[118] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSSx86: A full system simulator for x86 CPUs," in *Design and Automation Conference*, 2011.

[119] M. T. Yourst, "PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator," in *2007 IEEE International Symposium on Performance Analysis of Systems & Software*, 2007, pp. 23–34.

[120] F. Bellard, "QEMU, a fast and portable dynamic translator.," in *USENIX Annual Technical Conference, FREENIX Track*, 2005, vol. 41, p. 46.

[121] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, 2009, pp. 163–174.

[122] "MacSim." [Online]. Available: https://github.com/gthparch/macsim. [Accessed: 22-Oct-2018].

[123] G. F. Diamos, A. R. Kerr, S. Yalamanchili, and N. Clark, "Ocelot: a dynamic optimization framework for bulk-synchronous applications in heterogeneous systems," in *Proceedings of the 19th international conference on Parallel architectures and compilation techniques -*

PACT '10, 2010, p. 353.

[124] P. Abad, P. Prieto, L. G. Menezo, A. Colaso, V. Puente, and J.-Á. Gregorio, "TOPAZ: An Open-Source Interconnection Network Simulator for Chip Multiprocessors and Supercomputers," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, 2012, pp. 99–106.

[125] L. G. Menezo, V. Puente, and J.-A. Gregorio, "Flask coherence: A morphable hybrid coherence protocol to balance energy, performance and scalability," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 198–209.

[126] P. Abad, V. Puente, J. A. Gregorio, and P. Prieto, "Rotary router: an efficient architecture for CMP interconnection networks," in *Proceedings of the 34th annual international symposium on Computer architecture - ISCA '07*, 2007, p. 116.

[127] P. Prieto, V. Puente, and J. A. Gregorio, "CMP off-chip bandwidth scheduling guided by instruction criticality," in *Proceedings of the 27th international ACM conference on International conference on supercomputing - ICS '13*, 2013, p. 379.

[128] P. Abad, P. Prieto, V. Puente, and J.-A. Gregorio, "Improving last level shared cache performance through mobile insertion policies (MIP)," *Parallel Comput.*, vol. 49, pp. 13–27, Nov. 2015.

[129] P. Abad, P. Prieto, V. Puente, and J.-A. Gregorio, "AC-WAR: Architecting the Cache Hierarchy to Improve the Lifetime of a Non-Volatile Endurance-Limited Main Memory," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 1, pp. 66–77, Jan. 2016.

[130] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart, "LINPACK Users' Guide," *Siam*, 1979.

[131] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, "Introducing the Graph 500," *Cray User's Gr.*, 2010.

[132] H. Jin, M. Frumkin, and J. Yan, "The OpenMP implementation of NAS parallel benchmarks and its performance," *NAS Tech. Rep. NAS-99-011*, 1999.

[133] P. Luszczek *et al.*, "Introduction to the HPC Challenge Benchmark Suite," *Tech. Rep.*, 2005.

[134] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 44–54.

[135] J. A. S. C. Rodrigues *et al.*, "Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing," *IMPACT Tech. Rep.*, 2012.

[136] M. Burtscher, R. Nasre, and K. Pingali, "A quantitative study of irregular programs on GPUs," in *2012 IEEE International Symposium on Workload Characterization (IISWC)*, 2012, pp. 141–151.

[137] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, 2010, p. 143.

[138] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis," in *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*, 2010, pp. 41–51.

[139] G. Southern and J. Renau, "Deconstructing PARSEC scalability," in *Proceedings of the*

Annual Workshop on Duplicating, Deconstructing, and Debunking (WDDD), 2015.

[140] A. Mohammad, U. Darbaz, G. Dozsa, S. Diestelhorst, D. Kim, and N. S. Kim, "dist-gem5: Distributed simulation of computer clusters," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2017, pp. 153–162.

[141] H. Wu, F. Liu, and R. B. Lee, "Cloud Server Benchmark Suite for Evaluating New Hardware Architectures," *IEEE Comput. Archit. Lett.*, vol. 16, no. 1, pp. 14–17, Jan. 2017.

[142] J. D. Gindele, "Buffer Block Prefetching Method," *IBM Tech. Discl. Bull.*, vol. 20, pp. 696–697, 1977.

[143] J.-L. Baer and T.-F. Chen, "An effective on-chip preloading scheme to reduce data access penalty," in *Proceedings of the 1991 ACM/IEEE conference on Supercomputing - Supercomputing '91*, 1991, pp. 176–186.

[144] H. Radamson *et al.*, "The Challenges of Advanced CMOS Process from 2D to 3D," *Appl. Sci.*, vol. 7, no. 10, p. 1047, Oct. 2017.

[145] T. Ungerer and D. Fey, "Report on Disruptive Technologies for years 2020-2030," 2016.

[146] T. Yoshida *et al.*, "Sparc64 XIfx: Fujitsu's Next-Generation Processor for High-Performance Computing," *IEEE Micro*, vol. 35, no. 2, pp. 6–14, Mar. 2015.

[147] R. Smith, "The AMD Radeon R9 Fury X Review: Aiming for the Top," 2015. [Online]. Available: https://www.anandtech.com/show/9390/the-amd-radeon-r9-fury-x-review. [Accessed: 09-Oct-2018].

[148] J. Boukhobza, S. Rubini, R. Chen, and Z. Shao, "Emerging NVM: A Survey on Architectural Integration and Research Challenges," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 23, no. 2, pp. 1–32, Nov. 2017.

[149] O. Mutlu, "Main memory scaling: Challenges and solution directions," in *More than Moore Technologies for Next Generation Computer Design*, Springer, 2015, pp. 127–153.

[150] Y. Xie, "Modeling, Architecture, and Applications for Emerging Memory Technologies," *IEEE Des. Test Comput.*, vol. 28, no. 1, pp. 44–51, Jan. 2011.

[151] International Technology Roadmap of Semiconductors (ITRS 2.0), "Beyond CMOS," 2015.

[152] A. Makarov, V. Sverdlov, and S. Selberherr, "Emerging memory technologies: Trends, challenges, and modeling methods," *Microelectron. Reliab.*, vol. 52, no. 4, pp. 628–634, Apr. 2012.

[153] S. Tehrani *et al.*, "Magnetoresistive random access memory using magnetic tunnel junctions," *Proc. IEEE*, vol. 91, no. 5, pp. 703–714, May 2003.

[154] D. Kumar, R. Aluguri, U. Chand, and T. Y. Tseng, "Metal oxide resistive switching memory: Materials, properties and switching mechanisms," *Ceram. Int.*, vol. 43, pp. S547–S556, Aug. 2017.

[155] H.-S. P. Wong *et al.*, "Phase Change Memory," *Proc. IEEE*, vol. 98, no. 12, pp. 2201–2227, Dec. 2010.

[156] "Intel, STMicroelectronics Deliver Industry's First Phase Change Memory Prototypes," 2008. [Online]. Available: http://investors.micron.com/releasedetail.cfm?releaseid=467789. [Accessed: 23-Oct-2018].

[157] Y. Choi *et al.*, “A 20nm 1.8V 8Gb PRAM with 40MB/s program bandwidth,” in *2012 IEEE International Solid-State Circuits Conference*, 2012, pp. 46–48.

[158] B. Tallis, “Intel Launches Optane Memory M.2 Cache SSDs For Costumer Market,” 2017. [Online]. Available: https://www.anandtech.com/show/11227/intel-launches-optane-memory-m2-cache-ssds-for-client-market. [Accessed: 10-Oct-2018].

[159] M. Julliere, “Tunneling between ferromagnetic films,” *Phys. Lett. A*, vol. 54, no. 3, pp. 225–226, Sep. 1975.

[160] S. Mittal and J. S. Vetter, “A Survey of Software Techniques for Using Non-Volatile Memories for Storage and Main Memory Systems,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1537–1550, May 2016.

[161] S. Senni, L. Torres, G. Sassatelli, A. Bukto, and B. Mussard, “Exploration of Magnetic RAM Based Memory Hierarchy for Multicore Architecture,” in *2014 IEEE Computer Society Annual Symposium on VLSI*, 2014, pp. 248–251.

[162] H.-Y. Cheng *et al.*, “LAP: Loop-Block Aware Inclusion Properties for Energy-Efficient Asymmetric Last Level Caches,” in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 103–114.

[163] E. Kultursay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, “Evaluating STT-RAM as an energy-efficient main memory alternative,” in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 256–267.

[164] Everspin, “Everspin Announces Sampling of the World’s First 1-Gigabit MRAM Product,” 2017. [Online]. Available: https://www.everspin.com/sites/default/files/pressdocs/Everspin_Announces_Customer_Sampling_1Gb.pdf. [Accessed: 23-Oct-2018].

[165] “Researchers celebrate 20th anniversary of IBM’s invention of Spin Torque MRAM by demonstrating scalability for the next decade.” [Online]. Available: https://www.ibm.com/blogs/research/2016/07/ibm-celebrates-20-years-spin-torque-mram-scaling-11-nanometers/. [Accessed: 28-Sep-2018].

[166] R. Wiesendanger, “Nanoscale magnetic skyrmions in metallic films and multilayers: a new twist for spintronics,” *Nat. Rev. Mater.*, vol. 1, p. 16044, Jun. 2016.

[167] W. Kang, Y. Huang, X. Zhang, Y. Zhou, and W. Zhao, “Skyrmion-Electronics: An Overview and Outlook,” *Proc. IEEE*, vol. 104, no. 10, pp. 2040–2061, Oct. 2016.

[168] A. J. Annunziata *et al.*, “Racetrack memory cell array with integrated magnetic tunnel junction readout,” in *2011 International Electron Devices Meeting*, 2011, p. 24.3.1-24.3.4.

[169] K. v. Bergmann, “Magnetic bubbles with a twist,” *Science (80-.).*, vol. 349, no. 6245, pp. 234–235, Jul. 2015.

[170] J. C. Slonczewski, “Current-driven excitation of magnetic multilayers,” *J. Magn. Magn. Mater.*, vol. 159, no. 1–2, pp. L1–L7, 1996.

[171] L. Berger, “Emission of spin waves by a magnetic multilayer traversed by a current,” *Phys. Rev. B*, vol. 54, no. 13, pp. 9353–9358, Oct. 1996.

[172] A. Brataas and K. M. D. Hals, “Spin–orbit torques in action,” *Nat. Nanotechnol.*, vol. 9, no. 2, pp. 86–88, Feb. 2014.

[173] S. Emori, U. Bauer, S.-M. Ahn, E. Martinez, and G. S. D. Beach, “Current-driven dynamics of

chiral ferromagnetic domain walls," *Nat. Mater.*, vol. 12, no. 7, pp. 611–616, Jul. 2013.

[174] S. Bhatti, R. Sbiaa, A. Hirohata, H. Ohno, S. Fukami, and S. N. Piramanayagam, "Spintronics based random access memory: a review," *Mater. Today*, vol. 20, no. 9, pp. 530–548, Nov. 2017.

[175] S. Fukami *et al.*, "20-nm magnetic domain wall motion memory with ultralow-power operation," in *2013 IEEE International Electron Devices Meeting*, 2013, p. 3.5.1-3.5.4.

[176] Y. Zhang, W. S. Zhao, D. Ravelosona, J.-O. Klein, J. V. Kim, and C. Chappert, "Perpendicular-magnetic-anisotropy CoFeB racetrack memory," *J. Appl. Phys.*, vol. 111, no. 9, p. 093925, May 2012.

[177] A. J. Schellekens, A. van den Brink, J. H. Franken, H. J. M. Swagten, and B. Koopmans, "Electric-field control of domain wall motion in perpendicularly magnetized materials," *Nat. Commun.*, vol. 3, p. 847, May 2012.

[178] D. Chiba *et al.*, "Electric-field control of magnetic domain-wall velocity in ultrathin cobalt with perpendicular magnetization," *Nat. Commun.*, vol. 3, p. 888, Jun. 2012.

[179] T. Koyama *et al.*, "Current-induced magnetic domain wall motion below intrinsic threshold triggered by Walker breakdown," *Nat. Nanotechnol.*, vol. 7, no. 10, pp. 635–639, Oct. 2012.

[180] Y. Zhang *et al.*, "Ring-shaped Racetrack memory based on spin orbit torque driven chiral domain wall motions," *Sci. Rep.*, vol. 6, p. 35062, Oct. 2016.

[181] G. Wang *et al.*, "Ultra-Dense Ring-Shaped Racetrack Memory Cache Design," *IEEE Trans. Circuits Syst. I Regul. Pap.*, pp. 1–11, 2018.

[182] W. Kang *et al.*, "Complementary Skyrmion Racetrack Memory With Voltage Manipulation," *IEEE Electron Device Lett.*, vol. 37, no. 7, pp. 924–927, Jul. 2016.

[183] X. Chen *et al.*, "Complementary Skyrmion Racetrack Memory Enables Voltage-Controlled Local Data Update Functionality," *IEEE Trans. Electron Devices*, vol. 65, no. 10, pp. 4667–4673, Oct. 2018.

[184] P. Lai *et al.*, "An Improved Racetrack Structure for Transporting a Skyrmion," *Sci. Rep.*, vol. 7, p. 45330, Mar. 2017.

[185] H. T. Fook, W. L. Gan, I. Purnama, and W. S. Lew, "Mitigation of Magnus Force in Current-Induced Skyrmion Dynamics," *IEEE Trans. Magn.*, vol. 51, no. 11, pp. 1–4, Nov. 2015.

[186] C. Zhang *et al.*, "Hi-fi playback: Tolerating position errors in shift operations of racetrack memory," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture - ISCA '15*, 2015, pp. 694–706.

[187] S. Motaman and S. Ghosh, "Adaptive Write and Shift Current Modulation for Process Variation Tolerance in Domain Wall Caches," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 24, no. 3, pp. 944–953, Mar. 2016.

[188] A. Ranjan, S. G. Ramasubramanian, R. Venkatesan, V. Pai, K. Roy, and A. Raghunathan, "DyReCTape: A Dynamically Reconfigurable Cache using Domain Wall Memory Tapes."

[189] Z. Sun, X. Bi, A. K. Jones, and H. Li, "Design exploration of racetrack lower-level caches," in *Proceedings of the 2014 international symposium on Low power electronics and design - ISLPED '14*, 2014, pp. 263–266.

[190] H. Xu, Y. Alkabani, R. Melhem, and A. K. Jones, "FusedCache: A Naturally Inclusive,

Racetrack Memory, Dual-Level Private Cache," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 2, no. 2, pp. 69–82, Apr. 2016.

[191] Z. Sun, X. Bi, W. Wu, S. Yoo, and H. H. Li, "Array Organization and Data Management Exploration in Racetrack Memory," *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1041–1054, Apr. 2016.

[192] Haifeng Xu, Yong Li, R. Melhem, and A. K. Jones, "Multilane Racetrack caches: Improving efficiency through compression and independent shifting," in *The 20th Asia and South Pacific Design Automation Conference*, 2015, pp. 417–422.

[193] E. Atoofian and A. Saghir, "Shift-aware racetrack memory," in *2015 33rd IEEE International Conference on Computer Design (ICCD)*, 2015, pp. 427–430.

[194] R. Venkatesan, S. G. Ramasubramanian, S. Venkataramani, K. Roy, and A. Raghunathan, "STAG: spintronic-tape architecture for GPGPU cache hierarchies," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 253–264, Oct. 2014.

[195] S. S. P. Parkin, "Data in the Fast Lanes of Racetrack Memory," *Sci. Am.*, vol. 300, no. 6, pp. 76–81, Jun. 2009.

[196] R. Venkatesan *et al.*, "Cache Design with Domain Wall Memory," *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1010–1024, Apr. 2016.

[197] M.J. Charney and A.P. Reeves, "Generalized correlation based hardware prefetching," Technical Report EE-CEG-95-1, Cornell University, 1995.

[198] K. J. Nesbit and J. E. Smith, "Data Cache Prefetching Using a Global History Buffer," *IEEE Micro*, vol. 25, no. 1, pp. 90–97, Jan. 2005.

[199] G. B. Kandiraju and A. Sivasubramaniam, "Going the distance for TLB prefetching: an application-driven study," in *Proceedings 29th Annual International Symposium on Computer Architecture*, pp. 195–206.

[200] S. Mittal, R. Wang, and J. Vetter, "DESTINY: A Comprehensive Tool with 3D and Multi-Level Cell Memory Modeling Capability," *J. Low Power Electron. Appl.*, vol. 7, no. 3, p. 23, Sep. 2017.

[201] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "CACTI 5.1," Technical Report HPL-2008-20, HP Labs, 2008.

[202] Xiangyu Dong, Cong Xu, Yuan Xie, and N. P. Jouppi, "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.