Predictor de Prestaciones para la Planificación de Memoria en Clústers de Computadores

Mónica Serrano, Julio Sahuquillo, Houcine Hassan, Salvador Petit and José Duato [1]

Resumen—
El hardware de acceso a memoria remota (RMA) permite a una placa base de un clúster acceder directamente a la memoria instalada en una placa remota del mismo clúster. En trabajos recientes, esta característica se ha usado para ampliar el espacio direccionable de memoria de las placas seleccionadas, permitiendo un mejor equilibrio de los recursos de memoria principal entre las aplicaciones del clúster. Este método es mucho más económico que implementar un sistema de memoria totalmente compartida.

En este contexto, el planificador de memoria se encarga de encontrar una distribución adecuada de memoria local y remota que maximice las prestaciones y garantice una QoS mínima entre las aplicaciones. Hay que tener en cuenta el hecho de que cambiar de distribución de memoria es un proceso lento que implica varias placas, así pues el planificador de memoria tiene que asegurarse de que la distribución objetivo proporciona mejores prestaciones que la actual.

En este artículo, se diseña un predictor de prestaciones para encontrar la mejor distribución de memoria para un conjunto dado de aplicaciones ejecutándose en una placa de un clúster. El predictor usa contadores hardware sencillos para estimar el impacto en las prestaciones de las diferentes distribuciones de memoria. Los contadores hardware facilitan al predictor la información necesaria sobre el tiempo invertido en el procesador, el acceso a memoria y la red.

El modelo de prestaciones utilizado por el predictor se ha validado en un simulador de microarquitectura detallado usando benchmarks reales. Los resultados muestran que la desviación de la predicción nunca es mayor del 5% , siendo menor del 0,5% en la mayoría de los casos.

Palabras clave— **Clústers de computadores, Planificación de memoria, Asignación de memoria remota, Estimación de prestaciones.**

I. Introducción

DESDE su introducción, los clústers de computadores han mejorado sus prestaciones y han reducido sus costes de implementación con respecto a los supercomputadores. Hoy en día, es fácil encontrar muchas de estas máquinas en los primeros puestos de rankings de computadores de altas prestaciones como el TOP500 [1]. Esta transición ha sido posible gracias a que las técnicas de microarquitecturas avanzadas y las soluciones de interconexión que solo estaban disponibles para los supercomputadores han entrado en el mercado y se han hecho más accesibles, lo cual permite nuevas formas de mejorar las prestaciones de los diseños de clústers actuales manteniendo, o incluso reduciendo, sus costes.

[1]Departamento de Informática de Sistemas y Computadores (DISCA), Universidad Politécnica de Valencia, España, e-mail: monica@gap.upv.es, {jsahuqui, husein, spetit, jduato}@disca.upv.es.

Sin embargo, en tanto que las arquitecturas de los clústers están poco acopladas por diseño, no existe un entorno estándar que dé soporte al acceso a la memoria instalada en nodos remotos. Por tanto, para lidiar con aplicaciones de requieren grandes cantidades de memoria (ej., las bases de datos y los servicios a nivel de empresa, las aplicaciones de computación paralela a gran escala, etc.), los sistemas de clústers han de depender de soluciones lentas basadas en sistema operativo, como el intercambio a discos RAM remotos o implementar memoria compartida a nivel de software. Este hecho reduce las ventajas competitivas de este tipo de sistemas.

Hasta ahora, el hardware de Acceso a Memoria Remota (RMA) [2], que da acceso directo a un nodo determinado a memoria remota, estaba disponible solo para sistemas de supercomputadores como el BlueGeneL [3], el BlueGene/P [4], o el Cray XT [5]. No obstante, cada vez es más común encontrar estas implementaciones entre los clústers de computadores. Por ejemplo, el consorcio HyperTransport [6], que está compuesto por más de 60 miembros de las empresas (AMD, HP, Dell, IBM, etc.) y universidades más punteras, está extendiendo la tecnología Hypertransport, permitiendo el desarrollo del acceso a memoria remota en clústers de computadores.

Este trabajo se centra en un prototipo que implementa las extensiones de Hypertransport mencionadas arriba y cuyos nodos se comunican mediante una red de interconexión rápida. En este contexto, asumimos que el sistema operativo que se está ejecutando en los nodos ofrece la posibilidad de repartir la memoria entre los mismos, posibilitando la asignación de porciones de memoria remota a las aplicaciones locales.

Como las distintas regiones de memoria presentan latencias muy diferentes, las prestaciones de una aplicación determinada dependen directamente de cómo está distribuida entre dichas regiones la memoria que tiene asignada. Cada aplicación contribuye con sus prestaciones a las prestaciones totales, así pues se requiere un planificador de memoria que maximice las prestaciones globales. Este planificador ha de considerar tanto las características (i.e., latencia, ancho de banda) de las diferentes regiones de memoria como los requerimientos de memoria de todas las aplicaciones en ejecución. Por ejemplo, asignar un 25% de la memoria remota disponible a una aplicación de uso intensivo de memoria podría conducir a peores resultados que asignar toda la memoria remota a una

aplicación con buena localidad de cache. Para decidir como distribuir las distintas regiones de memoria entre las aplicaciones en ejecución, el planificador necesita información sobre las prestaciones esperadas para una distribución de memoria dada. Para obtener esta información se pueden diseñar dos soluciones: i) efectuar un *profiling off-line* de los benchmarks variando la distribución de memoria y ii) predecir dinámicamente las prestaciones de los benchmarks midiendo la utilización de los recursos del sistema durante su ejecución. La primera solución se desarrolló en un trabajo anterior [7], en el cual analizamos como afecta la distribución de memoria a las prestaciones de aplicaciones con diferentes requerimientos de memoria, y presentamos un algoritmo ideal de asignación de memoria (llamado SPP) que distribuía el espacio de memoria entre las aplicaciones maximizando las prestaciones globales. La generalización del SPP a un número n de aplicaciones se publicó en [8], donde también presentamos un algoritmo heurístico eficiente que aproximaba los resultados de prestaciones del SPP a la vez que reducía su complejidad computacional en un factor de $(n-1)!$. Ambos algoritmos tenían en cuenta un parámetro de calidad de servicio (QoS) para cada aplicación para garantizar los requerimientos mínimos de prestaciones.

A diferencia de esos trabajos, este artículo propone un predictor de prestaciones que proporciona la información requerida por el planificador de memoria. El objetivo principal del predictor de prestaciones es su uso por parte del planificador de memoria de cara a maximizar las prestaciones del sistema mientras se garantizan los requerimientos de QoS específicos. Para llevar a cabo las predicciones, se necesitan 3 muestras de ejecuciones, cada una de las cuales considera que el *working-set* del benchmark se almacena por completo en una de las 3 regiones de memoria (i.e., L, Lb o R). Usando estas muestras, se estiman las prestaciones de cualquier otra distribución de memoria.

El predictor propuesto funciona a mediante un novedoso modelo de prestaciones que se alimenta de contadores hardware sencillos (como los que están disponibles en la mayoría de procesadores actuales) que miden la distribución del tiempo de ejecución invertido en el procesador, la memoria y los recursos de red. Aunque este modelo se puede implementar en cualquier procesador, este trabajo se basa en ejecución emphen orden por razones de simplicidad. El modelo se ha validado comparando sus estimaciones con los valores de prestaciones obtenidos por una ejecución real de los benchmarks en el entorno de simulación Multi2Sim [9]. Los resultados muestran que el predictor dinámico es muy preciso, pues su desviación con respecto a los resultados reales siempre está por debajo del 5%, y es mucho menor en la mayoría de los casos.

El resto de este artículo se organiza como sigue. La Sección II describe el prototipo del sistema. La Sección III detalla nuestra propuesta de modelo de prestaciones. La Sección IV valida el modelo mediante la comparación de sus predicciones con los resultados de la simulación detallada ciclo a ciclo. La Sección V trata de las investigaciones anteriores relacionadas con este trabajo y, finalmente, la Sección VI presenta algunos comentarios finales.

II. Prototipo de Clúster

Se está trabajando en el prototipo de un clúster con las capacidades hardware/software requeridas junto a la Universidad de Heidelberg [2], en la cual han diseñado las tarjetas de conexión RMA. La máquina consta de de 64 placas, cada una de las cuales incluye 4 procesadores Opteron de 4 núcleos a 2.0GHz en un sistema NUMA de 4 nodos (1 procesador por nodo), y una memoria RAM de 16GB en cada placa. La conexión a las placas remotas está implementada mediante una interfaz HyperTransport regular [10] a la placa local y una interfaz *High Node Count* HyperTransport [11] a las placas remotas. Esta interfaz está conectada a las placas remotas por medio de tarjetas compatibles con HTX [12].

Cuando un procesador emite una instrucción *load* o *store*, la operación de memoria se remite al controlador de memoria del nodo que se encargue de dicha dirección de memoria. Las tarjetas de conexión RMA incluyen su propio controlador, el cual procesa los accesos a memoria remota. A diferencia de los controladores de memoria típicos, el controlador RMA no tiene bancos de memoria directamente conectados a él, sino que depende de los bancos instalados en placas remotas. Este controlador se puede reconfigurar para que los accesos a una determinada dirección de memoria se remitan a una placa seleccionada.

Puesto que el prototipo se halla en construcción, para llevar a cabo los experimentos y validar el modelo de prestaciones propuesto, el clúster se ha modelado usando Multi2Sim. Multi2Sim es un entorno de simulación para procesadores superescalares, multihilo y multinúcleo. Es un simulador de microarquitectura dirigido por ejecución de tipo *application-only*, que permite simular ciclo a ciclo la ejecución de múltiples aplicaciones sin arrancar un sistema operativo completo.

El sistema completo ha sido escalado para obtener tiempos de simulación razonables. El sistema escalado está compuesto por 2 placas, cada una con un sistema NUMA de 2 nodos tal y como se muestra en la Figura 1. Cada nodo incluye un procesador con caches privadas, su controlador de memoria y la memoria RAM asociada.

La Tabla I muestra las características del subsistema de memoria, en el cual las latencias y las organizaciones de cache se asemejan a las del prototipo real. Se ha asumido que las tarjetas de conexión RMA no tienen capacidad de almacenamiento interno. Asimismo, el protocolo de coherencia de Multi2Sim se ha extendido para

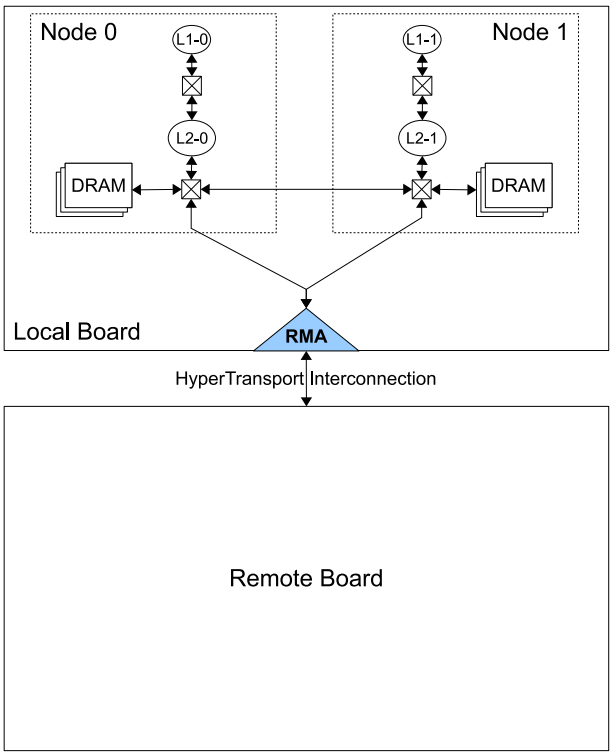

Fig. 1. Diagrama de Bloques del modelo de sistema NUMA y el RMA.

TABLA I
CARACTERÍSTICAS DEL SUBSISTEMA DE MEMORIA

Característica	**Descripción**
# de procesadores	2 por placa
Cache L1: tamaño, #vías, tamaño línea	64KB, 2, 64B
Latencia de cache L1	3
Cache L2: tamaño, #vías, tamaño línea	1MB, 16, 64B
Latencia de cache L2	6
Espacio direccionamiento memoria	512MB (256MB por placa)
Latencia L	100
Latencia Lb	142
Latencia R	410

modelar la funcionalidad del RMA.

III. Modelo de prestaciones

Un sistema cuyas aplicaciones en ejecución se pueden ejecutar usando diferentes distribuciones de memoria (L, Lb, R) necesita un mecanismo para determinar qué distribución de memoria se debería asignar a cada aplicación. Esta sección presenta una metodología para predecir el impacto de las diferentes distribuciones de memoria en las prestaciones, y el uso de las predicciones para guiar la asignación de las regiones de memoria a las aplicaciones de forma que se respeten las restricciones de memoria y se reduzca la pérdida de prestaciones.

Este trabajo asume que el predictor evalúa ocho posibles distribuciones de memoria (tres muestras y cinco casos estimados). Dicho número de datos elegidos del conjunto completo de distribuciones posibles de memoria define suficientemente las prestaciones de cada aplicación [8]. Para predecir las prestaciones (tiempo de ejecución) de una aplicación A en ejecución cuando se tiene una asignación de memoria $\{L = X,\ Lb = Y,\ R = Z\}$, se ha diseñado un método analítico.

Los procesadores actuales implementan contadores de prestaciones con vistas a la depuración que se pueden leer por software. En este artículo, estos contadores son utilizados por un mecanismo predictivo de asignación de memoria a las aplicaciones. Los contadores se usan para contar el número de ciclos invertidos en cada evento considerado durante un quántum de planificación completo.

A. Modelo analítico

El tiempo de ejecución T_{ej} de una aplicación dada se puede estimar a partir de dos componentes principales, tal y como se expone en la ecuación 1.

$$T_{ej} = C_{Dispatch} + C_{mem_stalls} \tag{1}$$

Cada C_x es el número de ciclos de procesador empleados en cierto tipo de actividad. Como se ha asumido que el ancho de la etapa *Dispatch* es 1, el tiempo de ejecución se puede expresar como la suma del número de instrucciones enviadas más el número de ciclos de parada debidos a accesos a memoria.

En el sistema diseñado, los ciclos de parada debidos a que la cola de *loads* y *stores* (LSQ) está llena son críticos para las prestaciones, principalmente en aquellos benchmarks que tienen una tasa alta de accesos a memoria. Por otra parte, la etapa *Dispatch* permanece parada durante la ejecución de una instrucción *load*. Esto incluye tanto los accesos a las caches privadas (i.e., L1 and L2) como a memoria principal, con sus respectivos tiempos de acceso así como los retrasos relacionados con la red o los riesgos estructurales.

Para estimar el IPC, el modelo de prestaciones clasifica los componentes de memoria del tiempo de ejecución en dependientes de la región de memoria e independientes de la región de memoria:

$$\begin{aligned} C_{mem_stalls} = {} & C_L + C_{Lb} + C_R \\ & + C_{private_caches} + C_{LSQ_iwidth} \end{aligned} \tag{2}$$

C_L, C_{Lb}, y C_R se refieren a los ciclos empleados en cada región de memoria, es decir, Local, Local a la Placa y Remota, respectivamente. Cada $C_{<region>}$ incluye los ciclos usados en diversas actividades relacionadas con esa región de memoria. En particular, se han tenido en cuenta los ciclos de parada debidos a las siguientes razones:

Tiempo de acceso a memoria principal. Este tiempo incluye tanto los ciclos consumidos en la lectura de datos de memoria principal como en el tráfico de mensajes a través de la red de memoria.

Delayed hit. Este tipo de parada ocurre cuando un acceso a memoria no puede llevarse a cabo porque el bloque accedido ya está bloqueado por

otra instrucción de memoria, es decir, un nuevo bloque se está trayendo.

Concurrencia de escritura. Esta parada sucede porque no se permite el acceso concurrente al mismo bloque en cierta cache si uno de ellos es de escritura.

LSQ Llena. La etapa de *Dispatch* se atasca porque no hay entradas libres en la LSQ.

Los componentes restantes de la ecuación se pueden considerar como una constante k para cada región de memoria. Los componentes independientes de la región son los que se listan abajo:

Tiempo de acceso a caches privadas. Número de ciclos empleados en el acceso al primer y segundo nivel de cache del sistema. Estos accesos son independientes de la región porque no se efectúan sobre ningún módulo de memoria principal.

Limitación por el ancho de la LSQ. Solo se puede emitir (etapa *Issue*) una *load* o *store* en un ciclo determinado. Por ese motivo, si una instrucción *load* está lista para ser emitida y hay un conflicto de acceso entre una *load* y una *store*, se emiten en orden de programa, y la instrucción más *joven* lo intentará de nuevo al siguiente ciclo.

La ecuación final usada por el predictor es 3:

$$T_{ej} = C_{Dispatch} + C_L + C_{Lb} + C_R + k \qquad (3)$$

B. Estimación de prestaciones

El modelo asume que la máquina en cuestión proporciona los contadores de prestaciones requeridos para obtener los valores de los componentes de la ecuación 3. Hay que destacar que el tráfico de la red se tiene en cuenta, por lo que la congestión también está cuantificada.

El predictor necesita ejecutar cada benchmark tres veces para recopilar los valores requeridos con el fin de estimar las prestaciones. Cada muestra equivale al caso de tener todos los accesos sobre una única región, es decir, *i)* todos los accesos en la región de memoria local (i.e. $T_{ej,L=100\%}$), *ii)* todos los accesos en la región de memoria del otro nodo de la placa local (i.e. $T_{ej,Lb=100\%}$), y *iii)* todos los accesos en la región de memoria remota (i.e. $T_{ej,R=100\%}$):

Muestra 1 (L = 100%, Lb = 0%, R = 0%): $T_{ej,L=100\%} = C_{L:L=100\%} + k$
Muestra 2 (L = 0%, Lb = 100%, R = 0%): $T_{ej,Lb=100\%} = C_{Lb:Lb=100\%} + k$
Muestra 3 (L = 0%, Lb = 0%, R = 100%): $T_{ej,R=100\%} = C_{R:R=100\%} + k$

Para estimar el tiempo de ejecución de una distribución de memoria dada, el predictor calcula un tiempo de ejecución ponderado, $T_{ej_ponderado}$, a partir de las tres muestras. Toma cada componente de región de memoria no nulo $C_{<region>}$ de las muestras y lo multiplica por la fracción $f_{<region>}$ de accesos a la región de memoria destino:

TABLA II
EJEMPLO DE FUNCIONAMIENTO DEL PREDICTOR DE PRESTACIONES

	$C_{<region>}$	$f_{<region>}$	$C_{<region>_pond}$
Muestra1	44687	0,5	22343,5
Muestra2	62236	0,5	31118
Muestra3	166757	0	0
k			2721346,3
$t_{ej_ponderado}$			2774807,8

$$\begin{aligned} T_{ej_ponderado} = {} & C_{L,L=100\%} \cdot (f_L) \\ & + C_{Lb,Lb=100\%} \cdot (f_{Lb}) \\ & + C_{R,R=100\%} \cdot (f_R) + k \end{aligned} \qquad (4)$$

Para cualquier distribución de memoria, se puede usar la ecuación 4 para predecir su tiempo de ejecución, dados los componentes de las tres muestras. Esto proporciona un mecanismo para identificar la distribución de memoria óptima para una determinada fase de ejecución con una pérdida mínima de prestaciones. Así pues, esta predicción formará parte de la entrada del planificador de memoria.

La Tabla II analiza un ejemplo de predicción para el benchmark FFT, en el cual el tiempo de ejecución de la distribución de memoria $(50\%, 50\%, 0)$ se obtiene a partir de las tres muestras. El tiempo de ejecución estimado es igual a 2774807,8 y el tiempo de ejecución de la simulación detallada ciclo a ciclo es 2774931, así pues el modelo ha obtenido una estimación que difiere menos del 0,005% del valor al que se aspira.

IV. VALIDACIÓN DEL MODELO

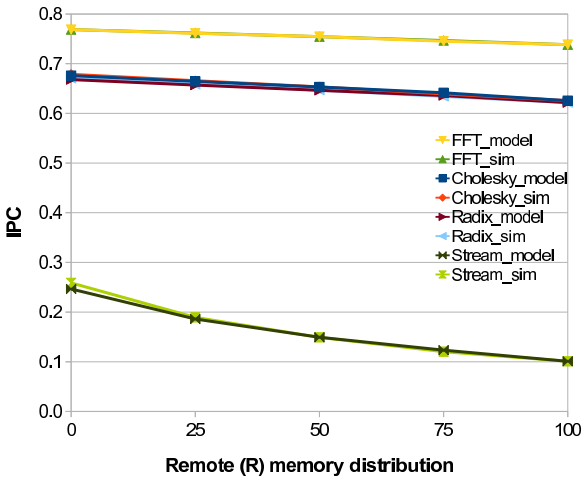

Fig. 2. Validación del modelo. Simulación detallada ciclo a ciclo *vs* modelo.

Esta sección analiza la precisión de la estimación. Hemos realizado experimentos para cuatro benchmarks con las ocho distribuciones de memoria: *i)*(100%, 0%, 0%), *ii)*(50%, 50%, 0%), *iii)*(0%, 100%, 0%), *iv)*(75%, 0%, 25%), *v)*(50%, 25%, 25%), *vi)*(50%, 0%, 50%), *vii)*(25%, 0%, 75%), *viii)*(0%,

0%, 100%). Posteriormente, hemos tomado los componentes de las tres muestras (*i*, *iii*, y *viii*) y hemos aplicado el modelo a cada benchmark para obtener el tiempo de ejecución para cada una de las distribuciones restantes. Finalmente, se han calculado las Instrucciones Por Ciclo (IPC) para cada caso.

La Figura 2 muestra la comparación de los resultados de prestaciones simulados (*sim*) contra los valores calculados por el predictor de prestaciones (*model*). La curva del modelo se solapa con la de la simulación detallada ciclo a ciclo, ya que el modelo presenta una desviación menor al 5% en el peor caso, estando cerca del 0% para algunos benchmarks, por ejemplo FFT.

V. Trabajo relacionado

Algunos trabajos de investigación anteriores han tratado el problema de la predicción de prestaciones para caracterizar y clasificar el comportamiento de memoria de las aplicaciones y predecir sus prestaciones.

Zhuravlev et al [13] estimaron que factores tales como el controlador de memoria, el bus de memoria y las contenciones del hardware de *prefetching* contribuyen más a la degradación de las prestaciones globales que la contención del espacio de cache. Para paliar el efecto de estos factores minimizaron el número total de fallos emitidos por cada cache. Para ello, desarrollaron algoritmos de planificación que repartían los hilos de ejecución de forma tal que la tasa de fallos se distribuía equitativamente entre las caches.

En [14] los autores proponen un algoritmo de clasificación para determinar comportamientos de compartición de caché entre los programas. Su esquema se puede implementar directamente en hardware para proporcionar una clasificación dinámica los distintos comportamientos. Proponen un esquema dinámico de particionamiento de cache muy simple que funciona ligeramente mejor que el esquema de Particionamiento de Cache *Utility-based* además de incurrir en menores costes de implementación.

En [15] se propone un modelo de prestaciones de cache compartida, rápido y preciso para procesadores multinúcleo. El modelo estima la degradación de prestaciones debida a la contención en la cache de los procesos que se ejecutan en CMPs. Utiliza histogramas de reuso de distancia, frecuencias de acceso a cache, y la relación entre el rendimiento y la tasa de fallos de cache de cada proceso. De esta manera se puede predecir su tamaño efectivo de cache cuando se ejecutan concurrentemente con otros procesos, permitiendo la estimación del rendimiento de las instrucciones. El error medio de la predicción de rendimiento del modelo fue del 1,57%.

En [16] los autores aplican técnicas de aprendizaje para predecir las prestaciones de procesadores multinúcleo. La contribución principal del estudio es la enumeración de atributos de programas que se ejecutan solos, los cuales se pueden usar para predecir las prestaciones de ejecuciones conjuntas. La ejecución conjunta implica la contención para los recursos compartidos entre las aplicaciones que se ejecutan concurrentemente.

Los artículos de investigación anteriores están enfocados a procesadores multinúcleo o CMP, sin embargo el trabajo propuesto se centra en los clústers de computadores para afrontar el problema de predecir el comportamiento de las aplicaciones que usan memoria remota y así permitir al planificador mejorar las prestaciones del sistema.

Otros artículos que hallamos en la bibliografía relacionados con la asignación de memoria remota se basan principalmente en el intercambio de memoria. Shuang et al. diseñan un sistema de paginación remota para la utilización de la memoria remota en clústers InfiniBand [17]. En [18], se describe el uso de memoria remota para intercambio de memoria virtual en clústers de computadores. Midorikawa et al. proponen un sistema de memoria distribuida (DLM), el cual es una solución software a nivel de usuario que proporciona una memoria virtual muy grande mediante el uso de memoria remota distribuida entre los nodos del clúster [19].

Estos trabajos usan la memoria remota para el intercambio de memoria entre los nodos del clúster y presentan su sistema como una mejora al intercambio con disco. Por el contrario, nuestra investigación pretende predecir las prestaciones del sistema dependiendo de las diferentes configuraciones de asignación de memoria remota entre las aplicaciones. Las predicciones se usarán por un planificador de memoria para decidir dinámicamente cuál es la mejor configuración y mejorar las prestaciones del sistema.

VI. Conclusiones

Este artículo ha presentado un predictor de prestaciones capaz de estimar el tiempo de ejecución para una determinada distribución de memoria de una aplicación. En primer lugar, desarrollamos un estudio para determinar los eventos que formarían parte de nuestro modelo, y los clasificamos como dependientes o independientes de la región de memoria. El modelo asume que el número de ciclos consumidos en cada evento en cuestión se obtiene a partir de los contadores hardware de la máquina utilizada.

El predictor diseñado se ha usado para estimar las prestaciones de distintas distribuciones de memoria para cuatro benchmarks. La precisión del predictor se ha validado, puesto que la desviación del modelo con respecto a los resultados reales es siempre menor del 5% y muy próxima al 0% en numerosos casos.

Este estudio constituye el primer paso de un trabajo más profundo en el campo de la planificación de memoria. Las prestaciones estimadas por el predictor alimentarán a un planificador de memoria que elegirá dinámicamente la distribución de memoria óptima para cada aplicación que se esté ejecutando concurrentemente en el sistema, de forma

que se alcancen las mejores prestaciones globales del mismo.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación (MICINN), el programa Consolider y los fondos de Plan E, así como por los fondos FEDER de la Comisión Europea, mediante las subvenciones a los proyectos CSD2006-00046 y TIN2009-14475-C04-01.

Referencias

[1] Hans Werner Meuer, "The top500 project: Looking back over 15 years of supercomputing experience," *Informatik-Spektrum*, vol. 31, pp. 203–222, 2008, 10.1007/s00287-008-0240-6.

[2] Mondrian Nussle, Martin Scherer, and Ulrich Bruning, "A Resource Optimized Remote-Memory-Access Architecture for Low-latency Communication," in *International Conference on Parallel Processing*, Sept. 2009, pp. 220–227.

[3] M. Blocksome, C. Archer, T. Inglett, P. McCarthy, M. Mundy, J. Ratterman, A. Sidelnik, B. Smith, G. Almási, J. Casta nos, D. Lieber, J. Moreira, S. Krishnamoorthy, V. Tipparaju, and J. Nieplocha, "Design and implementation of a one-sided communication interface for the IBM eServer Blue Gene®supercomputer," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, New York, NY, USA, 2006, p. 120, ACM.

[4] Sameer Kumar, Gábor Dózsa, Gheorghe Almasi, Philip Heidelberger, Dong Chen, Mark Giampapa, Michael Blocksome, Ahmad Faraj, Jeff Parker, Joe Ratterman, Brian E. Smith, and Charles Archer, "The deep computing messaging framework: generalized scalable message passing on the blue gene/P supercomputer," in *ICS*, 2008, pp. 94–103.

[5] V. Tipparaju, A. Kot, J. Nieplocha, M.T. Bruggencate, and N. Chrisochoides, "Evaluation of Remote Memory Access Communication on the Cray XT3," in *Parallel and Distributed Processing Symposium. IEEE International*, March 2007, pp. 1–7.

[6] "HyperTransport Technology Consortium. HyperTransport I/O Link Specification Revision 3.10, 2008. ," .

[7] M. Serrano, J. Sahuquillo, H. Hassan, S. Petit, and J. Duato, "A scheduling heuristic to handle local and remote memory in cluster computers," in *HPCC*. 2010, pp. 35–42, IEEE.

[8] M. Serrano, J. Sahuquillo, S. Petit, H. Hassan, and J. Duato, "A cost-effective heuristic to schedule local and remote memory in cluster computers," *The Journal of Supercomputing*, vol. 59, no. 3, pp. 1533–1551, 2012.

[9] R. Ubal, J. Sahuquillo, S. Petit, and P. López, "Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors," *Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing*, 2007.

[10] Chetana N. Keltcher, Kevin J. McGrath, Ardsher Ahmed, and Pat Conway, "The AMD Opteron Processor for Multiprocessor Servers," *IEEE Micro*, vol. 23, no. 2, pp. 66–76, 2003.

[11] J. Duato, F. Silla, and S. Yalamanchili, "Extending HyperTransport Protocol for Improved Scalability," *First International Workshop on HyperTransport Research and Applications*, 2009.

[12] H. Litz, H. Fröening, M. Nuessle, and U. Brüening., "A HyperTransport Network Interface Controller for Ultra-low Latency Message Transfers," *HyperTransport Consortium White Paper*, 2007.

[13] Sergey Zhuravlev, Sergey Blagodurov, and Alexandra Fedorova, "Addressing shared resource contention in multicore processors via scheduling," in *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2010, pp. 129–142.

[14] Y. Xie and G. H. Loh, "Dynamic Classification of Program Memory Behaviors in CMPs," *2nd Workshop on Chip Multiprocessor Memory Systems and Interconnects in conjunction with the 35th International Symposium on Computer Architecture*, 2008.

[15] Chi Xu, Xi Chen, Robert P. Dick, and Zhuoqing Morley Mao, "Cache contention and application performance prediction for multi-core systems," in *IEEE International Symposium on Performance Analysis of Systems and Software*, 2010, pp. 76–86.

[16] J.K. Rai, A. Negi, R. Wankar, and K.D. Nayak, "Performance prediction on multi-core processors," in *Computational Intelligence and Communication Networks (CICN), 2010 International Conference on*, nov. 2010, pp. 633 –637.

[17] Shuang Liang, Ranjit Noronha, and Dhabaleswar K. Panda, "Swapping to Remote Memory over InfiniBand: An Approach using a High Performance Network Block Device," in *CLUSTER*. 2005, pp. 1–10, IEEE.

[18] Paul Werstein, Xiangfei Jia, and Zhiyi Huang, "A Remote Memory Swapping System for Cluster Computers," in *Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2007, pp. 75–81.

[19] H. Midorikawa, M. Kurokawa, R. Himeno, and M. Sato, "DLM: A distributed Large Memory System using remote memory swapping over cluster nodes," in *IEEE International Conference on Cluster Computing*, October 2008, pp. 268–273.