Técnicas de Coscheduling y Herramientas de Monitorización para Clusters no Dedicados *

Francesc Solsona [1], Francesc Giné [1], Porfidio Hernández [2] y Emilio Luque [2]

[1]Departamento de Informática e Ingeniería Industrial, Universitat de Lleida, España.
{francesc,sisco}@eup.udl.es

[2]Departamento de Informática, Universitat Autònoma de Barcelona, España.
{p.hernandez,e.luque}@cc.uab.es

Resumen

En este trabajo presentamos el diseño e implementación de un cluster de PCs en un entorno PVM/Linux que ofrece un sistema dual computador paralelo - computador tradicional.

Nuestro proyecto está centrado en el desarrollo del software necesario para la construcción de máquinas paralelas de bajo costo, que nos permitan, a partir de componentes comerciales, abordar problemáticas de cómputo paralelo sin afectar demasiado la ejecución de las aplicaciones que se están ejecutando de forma rutinaria en el sistema.

Hemos implementado tanto técnicas de coscheduling explícito, implícito como dinámico, y analizado y comparado, tanto la eficiencia como el costo que suponen la implementación de los métodos comentados anteriormente mediante benchmarks standards. Los resultados obtenidos, medidos con herramientas que han sido desarrolladas a tal efecto, muestran la viabilidad de nuestras propuestas.

Palabras clave: NOW, cluster, Coscheduling, herramienta de monitorización, PVM

* Este trabajo ha estado financiado por la CICYT bajo contrato TIC98-0433

Técnicas de Coscheduling y Herramientas de Monitorización para Clusters no Dedicados

1 Introducción

Mucho se ha escrito, sobre la posibilidad de acercarnos a la potencia de cómputo que proporcionan los sistemas masivamente paralelos (MPP) utilizando componentes comerciales. Las diferentes técnicas conocidas que inciden en el problema han mostrado su utilidad y viabilidad, no obstante, en algunos casos, cuando se consideran complejas estructuras de comunicación y entornos operativos basados en middleware; las técnicas de coscheduling implícito o dinámico [1,2,3,4,5,6] forman parte todavía de cuestiones abiertas. Además, creemos en la viabilidad de la aplicación de dichas técnicas en entornos de cómputo paralelo de bajo costo. Las líneas de investigación que se abren abarca un amplio abanico de posibilidades de actuación.

Nuestra idea original se centraba en una doble línea de actuación:

- El diseño de una máquina virtual paralela (MVP) de bajo costo, usando a tiempo parcial los nodos de procesamiento mientras seguían ejecutándose las aplicaciones locales (o de usuario), con resultados aceptables de rendimiento tanto para las aplicaciones paralelas como para las locales.
- La creación de todo un entorno, construido a partir de un s/w base, que pudiese simplificar al usuario las tareas de programación, asignación, etc., ofreciéndole alternativas que pudiesen ser gestionadas/controladas de manera eficiente y totalmente transparente.

Frente a propuestas más generalistas, en cuanto a la funcionalidad que deberían cubrir los clusters creemos que, para poder ejecutar tanto aplicaciones paralelas como de usuario eficientemente es necesario ahondar en la planificación y sincronización de dichas aplicaciones de forma global. Una parte importante de nuestros esfuerzos, siguiendo esta línea, se orienta al desarrollo de modelos de rendimiento en relación a los middleware y manejadores de recursos utilizados, para derivar características de predecibilidad y desarrollar métricas de rendimiento (speedup, execution time, análisis de escalabilidad, etc.). A su vez, la monitorización contínua del sistema y su posterior análisis permitirán la inclusión de mecanismos de sintonización con el propósito de adaptar y afinar el sistema a los cambios de manera dinámica.

Una vez diseñado e implementado por nuestro grupo, el primer prototipo de MVP en un entorno PVM/Linux y probado su validez [7,8], mediante la aplicación de técnicas de coscheduling (dinámico, implícito) pretendemos afinar la MVP y extraer los parámetros significativos para construir los modelos de prestaciones. Igualmente, incluimos nuestras propias propuestas de planificadores mostrando su viabilidad mediante comparativas de rendimiento con otras propuestas de la literatura.

El resto del artículo está organizado como sigue. En la sección 2, se presentan las implementaciones basadas en técnicas de coscheduling explícito e implícito. En la sección 3 se desarrolla un modelo y algoritmo dinámico. En la sección 4, se presenta una herramienta de monitorización para evaluar los subsistemas de comunicación. En la sección 5, se evalúan las distintas técnicas de coscheduling mediante ejecuciones reales y simulación. Finalmente, se detallan las conclusiones y las líneas futuras de actuación.

2 Coscheduling Explícito e Implícito

En esta sección, se describen los métodos y métricas que permitirán obtener medidas de rendimiento de las técnicas de coscheduling explícito e implícito, sobre tareas distribuidas en un entorno NOW (o cluster) bajo PVM-Linux.

2.1 Coscheduling Explícito

La clave de la técnica basada en coscheduling explícito, consiste en planificar todas las tareas distribuidas en el cluster al mismo tiempo, y permitir que se ejecuten durante un periodo de tiempo. Desde un controlador global, se ejecuta un proceso en un nodo denominado master, éste controla los mensajes que son enviados (de forma broadcast) a cada proceso (denominado dts) que se ejecuta en cada uno de los nodos que componen el cluster, los cuales son los responsables de implementar el coscheduling explícito. Uno de los mensajes de control (init) informa a todos los procesos dts que deben empezar a enviar las señales de STOP y CONTINUE a los procesos distribuidos (de alta prioridad, residentes en su nodo) a intervalos regulares (ver también [8]). El tiempo empleado en arrancar todas las tareas distribuidas es:

$$T_{start} = W_s\ (local) + W_w\ (dts) + S_{sig}\ (CONT) + W_w\ (dis) + W_s\ (dts), \quad (1)$$

donde W_w/W_s es el tiempo transcurrido en despertar/suspender a dts, a una tarea local (local) o a una tarea distribuida (dis). S_{sig}(CONT) es el máximo tiempo transcurrido en enviar una señal CONTINUE a todas las tareas distribuidas en el nodo. El tiempo gastado en parar (T_{stop}) todas las tareas distribuidas es:

$$T_{stop} = W_s\ (dis) + W_w\ (dts) + S_{sig}\ (STOP) + W_w\ (local) + W_s\ (dts), \quad (2)$$

donde S_{sig} (STOP) es el tiempo máximo transcurrido en enviar una señal de STOP a todas las tareas distribuidas en el nodo. Puesto que el tiempo en entregar una señal a un grupo de procesos no depende del tipo de señal, consideramos que S_{sig} (STOP) = S_{sig} (CONTINUE) = S_{sig}. De la misma manera, los valores $W_w = W_s = W$ son considerados iguales. En consecuencia, (1) y (2) pueden reformularse como:

$$T_{ex} = T_{start} = T_{stop} = 4W + S_{sig} \quad (3)$$

2.2 Coscheduling Implícito

La técnica de coscheduling implícito se basa en planificar al mismo tiempo, solamente las tareas distribuidas que se están comunicando. Nosotros estamos interesados solamente en retrasar (mediante espera activa) las tareas distribuidas en espera de mensajes (antes de bloquearse) a lo más, durante un periodo igual al tiempo en realizar un cambio de contexto y no en retrasar la entrega de mensajes durante un round-trip como en [2, 3, 4], dado que las tareas distribuidas pueden adoptar múltiples patrones de comunicaciones, y los mensajes pueden llegar asíncronamente a las tareas distribuidas, en cualquier instante de tiempo. La métrica T_{im} sirve para calcular el máximo overhead añadido en el retraso, el cual nos proporcionará una primera referencia para elegir el intervalo de retraso (ir):

$$T_{im} = W_s\ (dis) + W_w\ (local) \quad (4)$$

3 Modelo de Coscheduling Dinámico (MCD)

En esta sección, formalizamos un modelo de coscheduling dinámico para un cluster no dedicado (denominado MCD). También formulamos un modelo para un algoritmo de coscheduling dinámico (SDCA: Share Dynamic Coscheduling Algorithm).

Algoritmo 1 muestra el pseudocódigo de un planificador apropiativo de tipo Round-Robin, para un sistema operativo de tiempo compartido de un nodo del cluster, asumido en el planificador global MCD. Otras funciones típicas del planificador (como salvar/restaurar el contexto de las tareas) que no influyen en nuestro modelo, no son consideradas. El planificador trabaja indefinidamente, mientras la Cola Ready (CR) de procesos preparados no esté vacía. Se supone que el planificador asigna la CPU a la primera tarea de la lista.
Cuando la tarea en ejecución finaliza, su rebanada de tiempo termina u otra tarea se apropia de la CPU debido a una decisión dinámica, la ejecución continúa en la línea 4 (etiqueta DISPATCH), para llevar a cabo la actualización de la información estadística de características dinámicas de dicha tarea después de ese punto.

Algoritmo 1 Planificador MCD en un nodo

```
1    do forever
2        if (número de procesos en CR ≠ NULL)
3          asignar  CPU al primer proceso de la CR;
4    DISPATCH:
5          almacenar información dinámica y de la CPU del proceso saliente;
6          mover el proceso saliente de la CPU al final de la CR;
7        endif;
8    enddo;
```

Algoritmo 2 muestra el pseudocódigo del algoritmo de coscheduling dinámico propuesto (SDCA), implementado en la función de librería “recibir” (pvm_recv) de PVM, el cual funciona en estrecha colaboración con el algoritmo de planificación local de cada nodo. En la sección INICIALIZACION se realizan las inicializaciones de gran parte de las variables globales del algoritmo. CODIGO RESIDUAL es el lugar donde reside el resto del código fuente de la rutina "recibir". Se asume que la recepción de un mensaje se realiza asíncronamente, por este motivo denominamos "async_receive" el punto de entrada a esa rutina. Notar que la rutina “recibir” solamente difiere de la original en el código añadido a partir de la línea 5. Las técnicas de coscheduling dinámico sólo se aplicarán cuando al mover un proceso, éste alcance la primera posición de la CR. En la línea 7, el proceso que recibe un mensaje, de acuerdo con la condición dinámica es reordenado (movido) en la CR. Las condiciones dinámicas aplicadas son:

· El número de mensajes recibidos; (un proceso puede adelantar a otro si tiene más mensajes pendientes en la Cola de Almacenamiento de Mensajes (CAM) que este último).

· El máximo número de adelantos sufridos; (un proceso no puede adelantar a otro si éste último ha alcanzado dicho máximo).

Algoritmo 2 Coscheduling dinámico (SDCA)

```
RUTINA Recibir
1       INICIALIZACION
2       async_receive (mensaje, proceso):
3           if (proceso no esta en la CR) insertar por la cola en CR (proceso); endif;
4       CODIGO RESIDUAL
5       marcar el proceso como potencialmente planificable;
6       if (proceso ≠ primero en la CR)
7           mover el proceso en la CR de acuerdo a las condiciones dinámicas;
8           incrementar el campo de adelanto de los procesos adelantados;
9         if (proceso alcanza la posición inicial en CR) Goto DISPATCH; endif;
10      endif;
```

4 Monito: La herramienta de monitorización

El comportamiento del sistema de comunicación es analizado, a partir de los datos proporcionados por una herramienta de monitorización implementada al efecto y denominada Monito. Monito muestra el estado de las colas del sistema implicadas en el proceso de comunicación, desde el nivel de aplicación (PVM), hasta el nivel físico (en el dispositivo físico de red o tarjeta de red).

Las colas de envío/recepción más importantes, a efectos de determinar overheads, overflows y cuellos de botella son: hosts, locltasks, pvmrxlist y txlist en PVM; write_queue/receive_queue en el nivel de socket; buffs/backlog y CBL/RFA en el dispositivo lógico y físico respectivamente.

El conjunto de utilidades implementadas son: dos servicios PVM, pvm_getpvmdstats y pvm_getaskstat; los módulos stadsoc, stadque y staddev; la llamada al sistema dev_queues y finalmente Netmon, una aplicación que monitoriza y recoge información acerca de estas utilidades.

4.1 Netmon

Una vez suministrados a la herramienta Netmon, los parámetros de periodo de muestreo (pm) en milisegundos y el tiempo de monitorización (tm) en segundos, la herramienta permite:

1.- Obtener información PVM.

(a) Obtener estadísticas del demonio "pvmd" de pvm. Es llevado a cabo por la llamada de pvm, pvm_getpvmdstats (ver Fig. 1 (izquierda)). La función pvm_getpvmdstats envía un mensaje del tipo TM_PVMDSTAT al demonio de PVM y espera respuesta (1). En el demonio se ha implementado una nueva función, tm_pvmdstat, para replicar a Netmon con otro mensaje TM_PVMDSTAT (2), conteniendo la información de las estructuras hosts y locltasks, como por ejemplo, los paquetes a entregar a los hosts remotos (en hosts) y paquetes a entregar a las tareas locales (en locltasks).

(b) Obtener estadísticas de las tareas PVM, mediante una nueva función pvm, pvm_getaskstat (ver Fig. 1 (dcha)). La función pvm_getaskstat envía un mensaje TM_TASKSTAT al demonio y espera respuesta de él (1). En el demonio, se ha incluido una nueva función para tratar esta nueva clase de mensajes, llamada

tm_taskstat. Esta función envía un mensaje del tipo TC_TASKINFO a todas las tareas pvm (2). Posteriormente, esta función espera el envío de un mensaje de reconocimiento de todas las tareas pvm generado por una nueva función pvm_tc_taskinfo (3) y envía un mensaje TM_TASKSTAT a Netmon (4). La información obtenida es el número y tamaño de los buffers de mensajes, esperando para enviar (recibir) en las colas txlist (pvmrxlist).

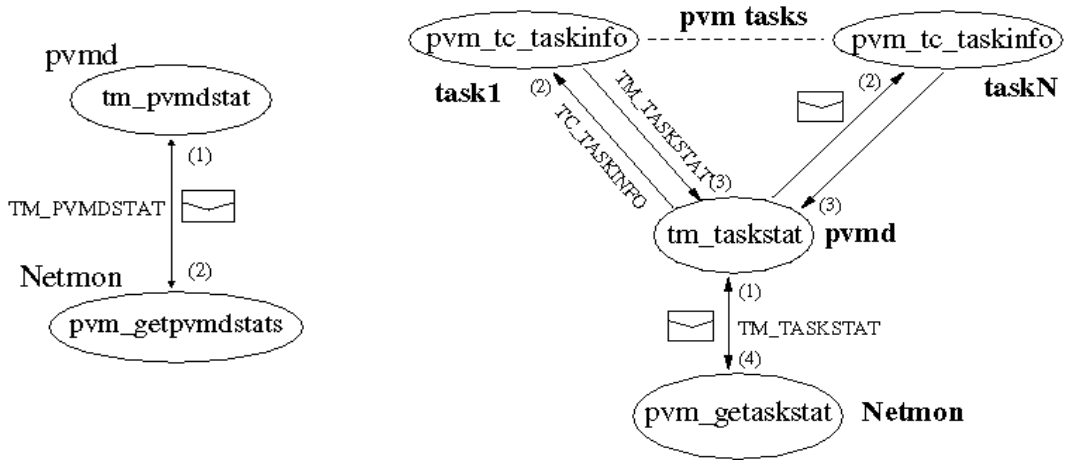

Figura 1: (izqda) monitorización de pvmd; (dcha) monitorización de tareas pvm

2.- Obtener información de Linux.

(a) Obtener estadísticas relacionadas con sockets. Netmon lee el fichero /proc/net/stadsoc, creado y mantenido por el módulo stadsoc para almacenar la información de la cola write_queue y receive_queue. La columna stadsoc de la Tabla 1, muestra la información proporcionada por este módulo. Notar que esta información es también suministrada por el kernel en tres diferentes ficheros /proc, pero el overhead en leerlos es inaceptable para pequeños periodos de muestreo. Ésta es la razón de implementar esta facilidad.

(b) Obtener estadísticas de los dispositivos lógicos. El método utilizado para obtener información de las colas de los dispositivos lógicos, backlog y buffs, es el mismo que en el módulo explicado anteriormente. Este módulo se denomina stadque y su fichero asociado es /proc/net/stadque (ver la columna stadque de la Tabla 1). No se conoce ninguna utilidad que permita obtener esta clase de información.

3.- Obtener información del dispositivo físico. El objetivo es capturar información adicional del dispositivo físico de red (ver la columna stadev de la Tabla 1), no soportada en el fichero /proc/net. Para muestrear las colas RFA y CBL, se ha implementado otro módulo, stadev. Su fichero asociado es /proc/dev/stadev.

Notar que la información de monitorización de PVM se obtiene mediante paso de mensajes; ésto puede producir algún overhead en el monitor. Cuando finaliza la monitorización, Netmon muestra en la pantalla el porcentaje de muestras que han sobrepasado el periodo de muestreo y el tiempo máximo extra requerido en el periodo de muestreo.

Stadsoc	Stadque	Stadev
Protocolo (tcp,udp o raw)	# de colas	# paquetes de colisión rec.
@IP y puerto fuente	longitud max. de la cola	# paquetes pendientes RFA
@IP y puerto destino	# sk_buff cola interactiva	# paquetes esperando trans.
# sk_buff en recv_queue	# sk_buff cola normal	# una colisión en trans.
# sk_buff en write_queue	# sk_buff cola background	# múltiples colisiones trans.
# bytes en recv_queue	# sk_buff cola backlog	# paquetes pendientes CBL
# bytes en write_queue		
# retransmisiones		

Tabla 1: Información de stadoc, stadque y staddev

5 Experimentación

La experimentación se ha realizado en una NOW (o cluster), formada por una red de comunicación Fast Ethernet de 100 Mbps y 4 PCs PVM-Linux con las mismas características: Pentium II a 350 Mhz, 128 MB de Ram y 512 KB de cache.

Se ha implementado una aplicación distribuida "sintree" a efectos de medir las prestaciones del entorno implementado. Su patrón de comunicaciones es de uno a varios y de varios a uno. También han servido como programas de pruebas dos benchmarks (clase A) del NAS: “is” y “mg” [9].

En una primera fase, se evalúa el correcto funcionamiento de la herramienta de monitorización. Posteriormente, se comparan las prestaciones de diferentes algoritmos de coscheduling explícito e implícito. Finalmente, se contrastan las prestaciones del algoritmo dinámico (expuesto en la sección 3) con otros algoritmos de coscheduling existentes por medio de simulación.

5.1 Evaluación de la herramienta Monito

En cada experimento se muestran: el modo de operación de PVM (RouteDirect o DontRoute [10]) y argumentos del benchmark sintree (# de procesos/tamaño de mensajes), por ejemplo, 32/8KB significa que los argumentos de sintree son 32 procesos y tamaño de los mensajes igual a 8KB. Los argumentos por defecto de Netmon son pm=100μs y tm=200s. En las figuras 2 y 3 se muestran los resultados más representativos del benchmark sintree, obtenidos en el nodo master.

5.1.1 Niveles físico, lógico y de socket

La Fig. 2 (izqda) muestra los resultados obtenidos en el dispositivo físico en el caso de 32/2MB. La cola CBL contiene por lo general un moderado número de paquetes (en el daemon los mensajes PVM se fragmentan en paquetes) transferidos desde los niveles superiores. La máxima capacidad de CBL y RFA es de 16 paquetes, pero por razones de seguridad, el driver siempre reserva dos elementos de CBL, por esta razón el máximo número que aparece en la Fig. 2 (izqda) es 14.

La Fig. 2 (dcha) muestra las estadísticas del nivel de socket para el caso 750/8KB. Notar, que la cola de recepción está saturada (la capacidad máxima es de 65.535 bytes). En los diferentes casos estudiados no ha habido saturación (o hechos relevantes) en los buffers de los demás niveles que implique su estudio y en consecuencia, no se muestran.

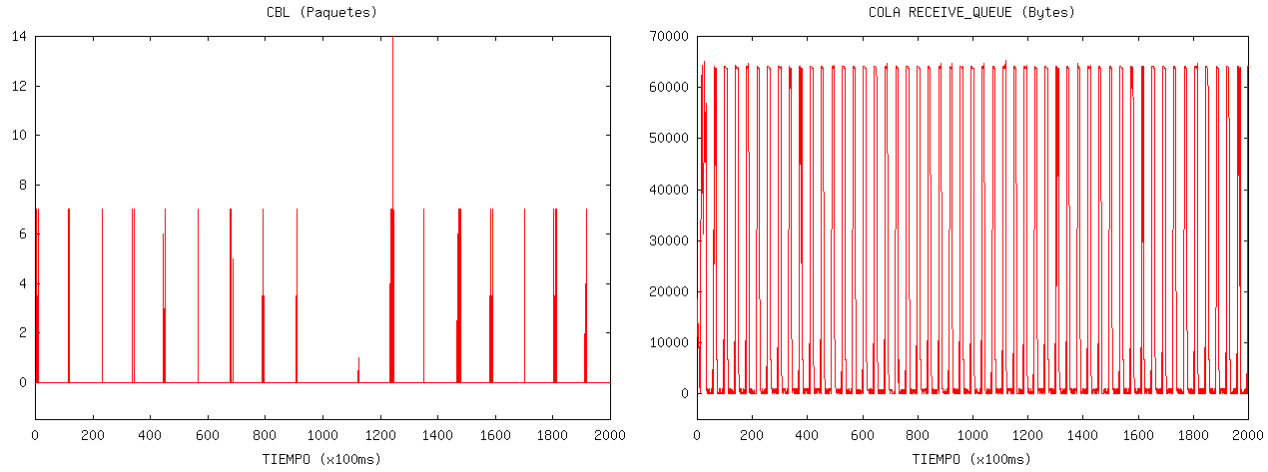

Fig. 2: DontRoute (izqda) paquetes en buffer de transmisión (cola CBL) para 32/2MB y (dcha) buffer de recepción socket en pvmd (receive_queue) para 750/8KB.

Estas gráficas confirman el correcto funcionamiento de Monito y su potencial capacidad para detectar los overheads, saturaciones y cuellos de botella del sistema de comunicaciones. Falta no obstante ahondar en su análisis y ampliar el Monito para MPI y comparar de esta forma los dos sistemas de paso de mensajes.

5.1.2 Nivel de PVM

La Fig. 3 muestra la cola de recepción (pvmrxlist) de la tarea master del benchmark sintree. Observar como el resultado de dividir la máxima capacidad alcanzada de pvmrxlist (=44MB) por el número total de paquetes (=22, número de cruces en un máximo de la figura) es 2MB (exactamente el tamaño de mensaje enviado); lo que vuelve a corroborar el correcto funcionamiento de Monito.

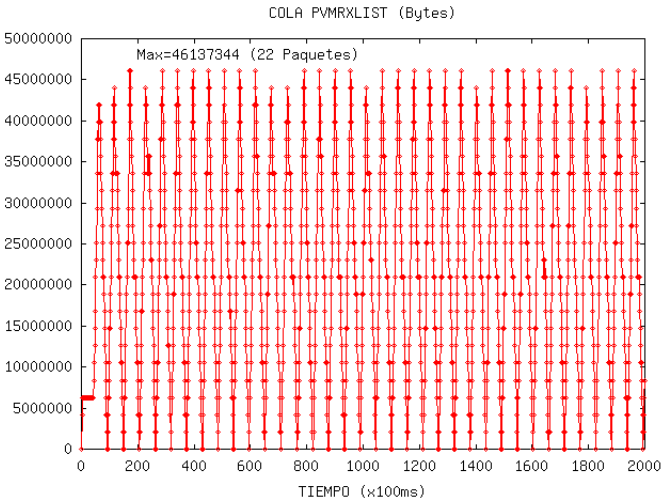

Fig. 3: RouteDirect cola pvmrxlist 25/2MB.

5.2 Experimentación considerando coscheduling explícito e implícito

Con objeto de comparar diferentes algoritmos de coscheduling, se han implementado varios entornos de experimentación. PVM: original. SPIN: coscheduling implícito (el intervalo de retraso

se realiza solo en la lectura de datos de un fragmento). MXI: la cabecera + los datos del fragmento se leen a la vez (en la implementación original, PVM lo realiza en dos pasos diferentes). MXISPIN: SPIN y MXI. PRIO: se asigna máxima prioridad a las tareas distribuidas. PRIOSPIN: PRIO y SPIN. EXPLICIT: periódicamente, transcurridos 90*ms*, el demonio dts de cada nodo envía una señal de STOP a todas las tareas distribuidas y después, pasados 10*ms*, dts envía otra señal de CONTINUE para despertarlas. Experimentalmente se obtuvo un valor de $T_{im} \cong 10$ μs, por lo que se eligió este valor para ir. En todos los experimentos, las comunicaciones entre tareas remotas se han realizado utilizando el modo PVM RouteDirect.

5.2.1 Prestaciones de las tareas distribuidas

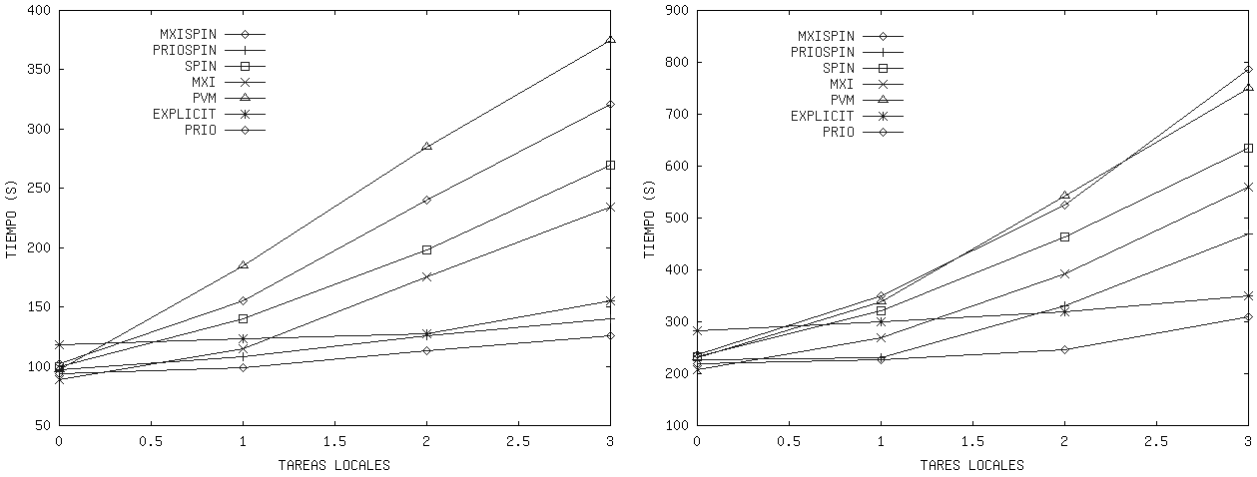

Fig. 4: Ejecución de sintree. (izqda) N=30.000. (dcha) N=70.000

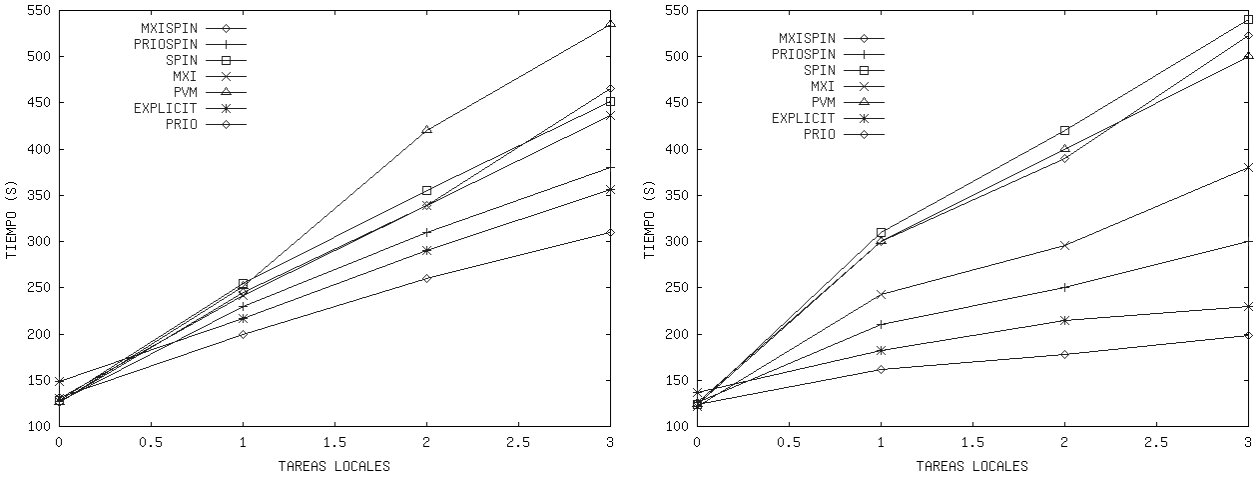

Figura 5: Ejecución de los benchmarks paralelos del NAS (izqda) mg y (dcha) is

La Fig. 4 muestra los tiempos de ejecución del algoritmo sintree, variando el número de iteraciones (N) del mismo, para los siete entornos citados anteriormente, mientras la carga local en cada nodo (simulada mediante aplicaciones de compilación) se varía de 0 a 3. Como cabía esperar, en el caso PRIO, el tiempo de ejecución de las tareas distribuidas es el mejor. El caso EXPLICIT sin tareas locales es la peor situación, pero a medida que se va incrementando la carga local, el rendimiento permanece constante, debido a que el tiempo asignado a las tareas distribuidas es independiente de

la carga local. El rendimiento de MXI y SPIN permanece siempre entre PVM y PRIO. SPIN es más rápido que PVM porque elimina el overhead que supone el bloqueo de las tareas distribuidas en la recepción de mensajes. El caso PRIOSPIN proporciona peores resultados que PRIO, por el "spinning" innecesario en este caso. MXISPIN trabaja peor que MXI, debido a que la penalización al finalizar las rebanadas de tiempo es mayor que el tiempo utilizado en realizar cambios de contexto.

La Fig. 5 muestra los resultados obtenidos en la ejecución de is y mg considerando los diferentes modelos de entornos. El comportamiento del benchmark mg es similar al sintree. Por otro lado, is, no trabaja tan bien como mg y sintree en el caso SPIN.

5.2.2 Prestaciones de las tareas locales

La influencia de los entornos diseñados en las tareas locales se midió mediante la métrica "Slowdown":

$$Sd_{MOD} = (TMODEL – TPVM) / TPVM * 100$$

donde TMODEL (TPVM) es el tiempo de ejecución de una tarea local ejecutada en dicho entorno (PVM original). Ver la Tabla 2.

Slowdown	PRIO	PRIOSPIN	EXPLICIT	SPIN	MXI	MXIPIN
Sintree	1.4	1.4	1.4	3.6	1.4	3.6
Is	2.8	2.8	4.2	2.1	0	2.1
Mg	90	92	42	8	1.6	8

Tabla 2: Slowdown de una tarea local de compilación

Como se esperaba, cuando se ejecuta aplicaciones con un gran volumen de comunicaciones, (sintree e is), el rendimiento de la tarea local decrece levemente. Por otro lado, se produce un slowdown importante si se ejecutan tareas fuertemente limitadas por cómputo (mg). En EXPICIT, la ejecución de las tareas distribuidas tiene un gran impacto en la tarea local y todavía más en el caso de PRIO y PRIOSPIN.

5.3 Experimentación MCD

Esta experimentación se ha realizado mediante simulación. Se han implementado tres programas diferentes. El primer programa simula el algoritmo SDCA, comentado en la sección 3. El segundo simula la técnica dinámica (DYN) [5,6]: en la recepción de un mensaje para la tarea l (t1), si executing_cpu_cycles(tl) + share < executing_cpu_cycles(te), se realiza un cambio de contexto a favor de la tarea l, donde "share" es una constante del sistema y "te" la tarea actualmente en ejecución. Finalmente, el último simula la técnica de spin-block (IMP) comentada en la sección 2.2.

A continuación se muestran los parámetros que se han tenido en cuenta para implementar los programas de simulación:

- Tiempo de llegada medio (mit): tiempo medio de llegada de tareas a la cola de tareas listas CR, simulada por medio de una distribución exponencial de media =mit.
- Tiempo medio de servicio: tiempo medio de servicio de una tarea (por la CPU). También, la función de densidad elegida es una exponencial con media=0.9.

- El número de tareas servidas: parámetro que marca el final de la simulación. Toda la experimentación ha sido realizada considerando este valor igual a 10000.
- Probabilidad de tarea distribuida (pdt): cada tarea generada es distribuida con probabilidad pdt, y local con probabilidad 1-pdt. La función de densidad es una Bernoulli con probabilidad pdt.
- Máximo número de mensajes (=5). Para cada tarea distribuida, se genera el número de mensajes recibidos. La función de densidad es discreta y uniforme en el intervalo [0 .. 5].

Generalmente, los mejores resultados se han obtenido para valores pequeños de spin y share. Por esta razón se han considerado valores de spin=0.01 y share=0.1.

La Fig. 6 muestra los resultados utilizando la métrica NodeCodeDe, definida como: la relación entre las tareas con mensajes en la CAM planificadas y todas las potencialmente planificables (planificadas y no planificadas) en un nodo. Para cada algoritmo, se muestra entre paréntesis, la probabilidad pdt.

Los buenos resultados obtenidos demuestran la efectividad de SDCA en el aprovechamiento del coscheduling potencial de las aplicaciones distribuidas.

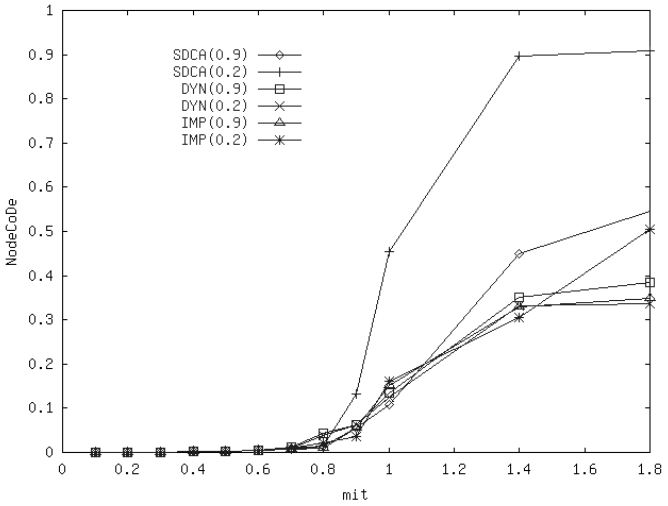

Fig. 6: Comparación de entornos utilizando la métrica NodeCodeDe. (pdt).

6 Conclusiones y trabajo futuro

Se ha presentado una propuesta viable de procesamiento a bajo costo, un entorno NOW o cluster construido a base de nodos Linux + entornos de paso de mensajes específicos (en nuestro caso PVM), construyendo una MVP con una doble funcionalidad: ejecución eficiente de aplicaciones paralelas garantizando al mismo tiempo la ejecución de la carga local en tiempos razonables. Se ha probado su viabilidad comparando el comportamiento de diferentes técnicas de coscheduling.

Igualmente, se ha presentado una herramienta de monitorización, Monito, que nos permite analizar las colas de mensajes implicadas en el proceso de comunicación desde el nivel de aplicación hasta el nivel físico.

Nuestro trabajo a corto plazo pretende incorporar un mayor número de nodos en el cluster a efectos de estudiar problemáticas de escalabilidad, broadcasting, cambios de contexto, permitir la incorporación en el entorno de múltiples aplicaciones paralelas y mejorar los modelos de representación de las cargas locales de los nodos. El trabajo futuro se centrará en encontrar nuevas métricas de eficiencia y escalabilidad que informen sobre el rendimiento global del sistema (tanto de aplicaciones paralelas como locales) y finalmente, ampliar Monito para un entorno MPI y comparar su rendimiento con PVM.

Referencias

[1] Ousterhout, J.K.: " Scheduling Techniques for Concurrent Systems". In Third International Conference on Distributed Computing Systems. (1982) 22-20.

[2] Arpaci, R.H., Dusseau, A.C., Vahdat, A.M., Liu, L.T., Anderson, T.E. and Paterson, D.A.: "The Interaction of Parallel and Sequential Workloads on a Network of Workstations". In Proc. of the ACM SIGMETRICS´95/ PERFORMANCE´95 Joint International Conference on Measurement and Modeling of Computer Systems. (1995) 267-278.

[3] Arpaci, R.H., Dusseau A.C., Culler, D.E. and Mainwaring, A.M.: "Scheduling with Implicit Information in Distributed Systems" In Proc. of the ACM SIGMETRICS´98/ PERFORMANCE´98 Joint International Conference on Measurement and Modeling of Computer Systems. (1998).

[4] Dusseau, A.C., Arpaci, R.H. and Culler, D.E.: "Effective Distributed Scheduling of Parallel Workloads". SIGMETRICS´96 Conference on Measurement and Modeling of Computer Systems. (1996).

[5] Sobalvarro, P.G. and Weihl, W.E.: "Demand-Based Coscheduling of Parallel Jobs on Multiprogrammed Multiprocessors". In Proc. of the IPPS´95 Workshop on Job Scheduling Strategies for Parallel Processing. (1995) 63-75.

[6] Sobalvarro, P.G., Pakin S., Weihl W.E. and Chien, A.: "Dynamic Coscheduling on Workstation Clusters". In Proc. of the IPPS´98 Workshop on Job Scheduling Strategies for Parallel Processing. (1998).

[7] Solsona, F., Giné, F., Hernández, P. and Luque, E.: “DTS: Un Entorno de Planificación Distribuido de Bajo Coste". IV CACIC. (1998) 695-706.

[8] Solsona, F., Giné, F., Hernández, P. and Luque, E.: "Implementing and Analysing an Effective Explicit Coscheduling Algorithm on a NOW". In Proc. of the VECPAR’2000. (2000).

[9] Bailey, D. et al.: " The NAS Parallel Benchmarks". International Journal of Supercomputer Applications. Vol. 5. nº 3. (1991).

[10] Geist, Al. et al.: "PVM: Parallel Virtual Machine – A User’s Guide and Tutorial for Network Parallel Computing". The MIT Press. (1994).