Efficient Buffer Usage Through a New Flow-Control Mechanism

Javier Prades, Federico Silla, José Duato,[1] Holger Fröning y Mondrian Nüessle[2]

Resumen— **En este artículo presentamos un detallado estudio sobre como diseñar un eficiente y escalable mecanismo de control de flujo. Analizando su comportamiento utilizando la implementación de MPI sobre la arquitectura EXTOLL. Hemos diseñado un control de flujo dinámico basado en créditos capaz de detectar el patrón de comunicaciones de una aplicación paralela y adecuar los recursos de buffering, por medio de una redistribución de los créditos, al patrón de comunicación. Nuestro análisis de prestaciones, realizado sobre un conjunto de pruebas de IMB, muestra que este nuevo mecanismo de control de flujo obtiene un gran nivel de eficiencia usando una pequeña cantidad de recursos de buffering.**

Palabras clave— **EXTOLL, VELO, RMA, MPI, control de flujo estático, control de flujo dinámico.**

I. Introducción

En este artículo presentamos un eficiente mecanismo de control de flujo para una emergente capa de red. Esta nueva capa de red forma parte del sistema de interconexión EXTOLL [1], el cual recientemente ha evolucionado desde una primera etapa de prototipado a una red de altas prestaciones disponible comercialmente, cuyo segmento de mercado es el HPC. EXTOLL es una innovadora solución de comunicación que provee latencias extremadamente bajas y evita el uso de costosos switches externos, pudiendo ser comparado con las versiones más rápidas de InfiniBand [2] pero a un coste mucho más económico. A fin de mostrar la extraordinaria escalabilidad y eficiencia de nuestro nuevo mecanismo de control de flujo, centramos nuestra presentación en su comportamiento sobre la implementación de MPI [3] sobre EXTOLL, basada en OpenMPI [4]. Sin embargo, otras implementaciones, como GASNet [5], podrían también ser usadas, ya que recientemente ha sido completada su implementación sobre EXTOLL [6].

El resto del artículo es organizado de la siguiente forma. En la sección II introduciremos las principales características de EXTOLL y mostraremos los detalles más destacables de la implementación MPI sobre EXTOLL, así como una breve descripción del cluster en el que hemos realizado las pruebas. Después de esto, las secciones III y IV introducirán los mecanismos de control de flujo y presentarán un primer mecanismo de control de flujo estático, el cual será usado como base para el desarrollo de nuestro nuevo mecanismo de control de flujo dinámico, que será presentado en la sección V y analizado en la sección VI. Finalmente la sección VII mostrará las principales conclusiones del artículo.

[1]Departament d'Informàtica de Sistemes i Computadors, Univ. Politècnica de València, e-mail: japraga@gap.upv.es, {fsilla,jduato}@disca.upv.es

[2]Computer Architecture Group, Univ. of Heidelberg, e-mail: {froening,nuessle}@uni-hd.de

II. MPI sobre EXTOLL

A. Arquitectura EXTOLL

EXTOLL es una arquitectura de interconexión para redes de altas prestaciones y provee soporte para la computación de altas prestaciones. La principal ventaja de EXTOLL es que proporciona una la latencia extremadamente baja, maximizando la tasa de mensajes entregados y la escalabilidad.

Así, EXTOLL optimiza las comunicaciones de mensajes cortos mediante VELO *(Virtualized Engine for Low Overhead)* [11]. Los mensajes VELO provenientes de cualquier emisor se almacenan en un único buffer de recepción existente en todos los receptores llamado *mailbox*. Por tanto, en el lado emisor debe existir una correcta información del espacio disponible en el lado receptor. Esta información es la que proporciona nuestro mecanismo de control de flujo. Para las transferencias de gran tamaño es utilizado RMA *(Remote Memory Access)* [12] que libera a la CPU casi completamente de las cargas de comunicación permitiendo un gran solapamiento y una baja sobrecarga, este tipo de transferencias se realizan a través de un protocolo *rendez-vous* siendo innecesario un mecanismo de control de flujo en este tipo de transferencias.

B. Implementación de MPI sobre EXTOLL

La implementación de MPI sobre EXTOLL está basada en la popular implementación OpenMPI. Por debajo del nivel MPI dos APIs *libvelo* y *librma* proporcionan directamente acceso a las funcionalidades de VELO y RMA respectivamente.

MPI usará VELO para el envío de mensajes pequeños, hasta un umbral de 2KB y RMA para transferencias mayores a este umbral. El tamaño máximo de un paquete VELO es de 64 bytes. Por lo tanto, para realizar envíos mayores a este tamaño se utilizarán tantos paquetes VELO como sean necesarios.

C. Cluster de 64 nodos con la arquitectura EXTOLL

Tanto el desarrollo como el análisis de prestaciones de nuestro mecanismo de control de flujo ha sido realizado en un cluster formado por 64 nodos con un total de 1024 cores e interconectado mediante la arquitectura EXTOLL.

Cada uno de los nodos incorpora 4 procesadores Quad-Core AMD Opteron 8350 [10] y 16 GB de memoria, además incorpora una tarjeta HTX [9] diseñada en la Universidad de Heidelberg que es la

responsable de la comunicación de los diferentes nodos del cluster.

III. Sobre los mecanismos de control de flujo

Básicamente, un mecanismo de control de flujo permite que receptores lentos, o receptores que están ocupados realizando otras tareas, no sufran un desbordamiento en sus buffers de recepción debido a emisores que transmiten gran cantidad de paquetes de datos. Nótese que si los buffers de recepción tuviesen un tamaño ilimitado, los problemas de desbordamiento no existirían y por lo tanto los mecanismos de control de flujo no serían necesarios. Sin embargo, estos buffers ilimitados consumirían grandes cantidades de memoria en algo que no es la aplicación en sí misma sino que forma parte de la infrastructura de comunicaciones y cuyo coste debe ser el más bajo posible. Además, estos recursos de memoria son extremadamente caros y, lo que es peor, dependientes del número de procesos involucrados.

Por lo tanto, los mecanismos de control de flujo pueden ser vistos como una técnica que permite establecer un límite máximo en los recursos de buffering utilizados por la capa de comunicación. Sin embargo, este límite tendrá un coste en el tiempo de ejecución de la aplicación, debido a dos causas: primero, algunos procesos pueden pararse mientras están enviando, debido al llenado de los buffers de recepción en el lado receptor. Segundo, el protocolo de control de flujo introduce una sobrecarga computacional así como un uso adicional del ancho de banda de la red debido a la necesidad de comunicar a los emisores el estado de los buffers de recepción.

Con el fin de implementar un mecanismo de control de flujo ligero y eficiente, hemos empezado con un control de flujo estático que después ha sido evolucionado en un control de flujo dinámico. Ambos protocolos están basados en el uso de créditos para mantener la información de las ranuras libres en los buffers de recepción, sin embargo, la versión dinámica maneja los créditos de una forma más flexible.

IV. Control de flujo estático basado en créditos

Para realizar la versión estática del control de flujo hemos iniciado nuestro desarrollo basándonos en el comúnmente utilizado *mecanismo de control de flujo basado en créditos*, así que nuestra implementación es una generalización de este mecanismo para adaptarlo a nuestra red de interconexión.

A. Principales adaptaciones

El mecanismo original de control de flujo basado en créditos establece que cada emisor dispone de cierto espacio en los buffers de recepción de los receptores. La cantidad exacta de estos recursos de buffering es indicada al inicio por el número de créditos. De esta forma, un receptor tiene tantos buffers independientes (o particiones de buffers) como emisores existen. Sin embargo, en nuestra red de interconexión un receptor únicamente dispone de un sólo buffer, llamado mailbox, el cual es compartido por todos sus emisores. Por lo tanto, un cambio trivial nos permitirá adaptar nuestro monolítico mailbox a la filosofía de créditos. Simplemente tendremos que dividirlo en tantas particiones independientes como posibles emisores existan. Nótese que el mailbox es dividido en varias porciones desde un punto de vista lógico, pero no de forma física. Por tanto, paquetes de datos procedentes de diferentes emisores no necesariamente serán almacenados en diferentes particiones del mailbox, sino que estarán mezclados. Nótese que el espacio del mailbox ha sido dividido de forma lógica pero se ha hecho una distribución física del espacio. De esta forma, seguimos teniendo un solo puntero de lectura y de escritura.

Otro cambio con respecto al protocolo original de control de flujo basado en créditos está relacionado con la actualización de la cuenta de créditos en los emisores. En el mecanismo original, cuando un receptor libera una ranura de los buffers de recepción, un crédito es devuelto al emisor correspondiente. Sin embargo, este proceso genera gran cantidad de tráfico por lo que muchas implementaciones [13] de este control de flujo acumulan varios créditos con la finalidad de devolver un único paquete que no consuma tantos recursos de la red. En nuestra red de interconexión el único método de comunicación entre procesos es mediante el intercambio de mensajes, por lo que un receptor, cuando recupere créditos, los irá acumulando hasta cierto umbral y una vez alcanzado generará un paquete que será devuelto al emisor para que actualice su cuenta de créditos. De ahora en adelante, nos referiremos a este tipo de paquetes como *paquetes de control* porque son generados por el mecanismo de control de flujo. De forma similar, nos referiremos a los paquetes de datos regulares, generados por la aplicación paralela, simplemente como *paquetes de datos*.

B. Funcionamiento del control de flujo estático

Con el objetivo de evitar posibles confusiones y dada la dualidad de funciones emisor/receptor que puede darse en cualquiera de los procesos, nos referiremos a un proceso como *emisor* cuando está enviando paquetes de datos a otros procesos y está extrayendo de su mailbox paquetes de control para proseguir con el envío y como *receptor* cuando está extrayendo de su mailbox paquetes de datos procedentes de otros emisores y retornando los paquetes de control que se originan al alcanzar el umbral de retorno de créditos.

Inicialmente dividimos el mailbox de cada proceso en dos regiones, una para almacenar paquetes de datos, *región de datos*, provenientes de los distintos emisores y otra para almacenar paquetes de control, *región de control*, provenientes de los distintos receptores. La región de datos será repartida entre los distintos procesos de forma equitativa, de este modo, después de la repartición, cada proceso, al realizar funciones de emisor, dispondrá de un número

de ranuras en el mailbox de cada receptor. Cada una de estas ranuras puede albergar un mensaje VELO. Por lo tanto, cada una de las ranuras equivaldrá a un crédito y la porción de la región de datos correspondiente a cada emisor, será pues su *cuota de créditos*. Por otro lado la región de control también será repartida entre los distintos procesos de forma equitativa, de este modo cada proceso, al realizar funciones de receptor, dispondrá de un número de ranuras en el mailbox de cada emisor, donde podrá mandar los mensajes de control. La porción correspondiente a cada receptor será pues su número de *ranuras de control*. Finalmente estableceremos el valor *umbral de retorno de créditos* según la ecuación:

$$(1)\ umbral=(cuota\_cred\ div\ (ranuras\_ctrl+1))+1$$

De este modo los paquetes de control siempre podrán ser enviados hacia el emisor, ya que el receptor, al alcanzar el umbral de retorno de créditos, sabrá que en el emisor existe al menos una ranura de control libre. Esto es debido a que el umbral establecido en la ecuación 1 garantiza que el emisor tenga que utilizar créditos contenidos en los paquetes de control y por tanto liberarán ranuras de control, para que en el receptor se alcance el umbral. Por último añadimos una restricción al tamaño de la cuota de créditos de los emisores, el tamaño mínimo de la cuota de créditos deberá ser:

$$(2)\ cuota\_creditos\_min=ranuras\_ctrl+1$$

Garantizando que el umbral de retorno de créditos será siempre mayor o igual a 2.

Cuando la etapa de inicialización ha finalizado, el funcionamiento del control de flujo estático es idéntico al del mecanismo original: cuando sea que un paquete de datos es enviado hacia un buffer de recepción, la cuenta de créditos del emisor es decrementada. Si la cuenta de créditos alcanza el valor cero, esto significa que no hay ranuras disponibles en el buffer receptor y, por lo tanto, no podrán ser enviados más paquetes de datos. Por otro lado, cuando un receptor libera una ranura de su buffer de recepción, un crédito es recuperado y en consecuencia la cuenta de créditos recuperados es incrementada. Cuando esta cuenta alcanza el valor umbral de retorno de créditos, un paquete de control es enviado hacia el emisor y entonces la cuenta de créditos recuperados es reiniciada. Nótese que los paquetes de control no consumen créditos al ser enviados y tampoco incrementan la cuenta de créditos recuperados al ser extraídos del mailbox. Este tipo de paquetes son controlados automáticamente por la ecuación 1 y la región de control reservada en el mailbox.

C. Inconvenientes del control de flujo estático

El control de flujo estático presentado en la sección anterior presenta serios inconvenientes. Un particionado estático del mailbox es muy adecuado cuando el patrón de comunicación está extremadamente balanceado, es decir, cuando todos los procesos intercambian mensajes de datos directamente con todos los demás procesos. Desafortunadamente, las aplicaciones paralelas no siempre siguen un patrón tan balanceado. La mayoría de las veces un proceso únicamente intercambia mensajes con un pequeño porcentaje del resto de procesos y esta situación se hace más evidente cuando el número de procesos implicados es muy grande, ya que la mayoría de las operaciones colectivas de MPI están optimizadas para crear árboles de comunicación que reducen considerablemente el número de comunicaciones entre procesos [8], provocando que el tiempo de ejecución se reduzca considerablemente. En estas situaciones un particionado estático del mailbox no es eficiente porque los recursos de buffering que no son usados por un emisor no pueden ser entregados a otros emisores que si los requieren, debido a la naturaleza estática del particionado. En otras palabras, todos los procesos tienen asignada la misma porción del mailbox, independiéntemente de su actividad, causando que procesos con mucha actividad no puedan usar los recursos que otros procesos no usan.

Esta situación es mostrada en la Figura 1, donde la sobrecarga debida al control de flujo es mostrada para cuatro diferentes patrones de comunicación:

1. 1024 procesos ejecutan una operación alltoall que los involucra a todos. Un proceso dado se comunica con todos los restantes
2. 1024 procesos ejecutan dos operaciones simultáneas alltoall que involucran a 512 procesos cada una. Un proceso dado únicamente se comunica con el 50% de los restantes procesos
3. 1024 procesos ejecutan cuatro operaciones simultáneas alltoall que involucran a 256 procesos cada una. Un proceso dado únicamente se comunica con el 25% de los restantes procesos
4. 1024 procesos ejecutan ocho operaciones simultáneas alltoall que involucran a 128 procesos cada una. Un proceso dado exclusivamente se comunica con el 12.5% de los restantes procesos

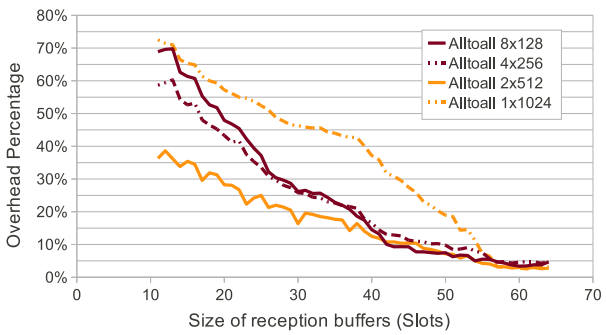

Fig. 1. Sobrecarga inducida por el control de flujo estático para cuatro patrones de comunicación distintos

Nótese que en ninguno de los casos casos anteriores han sido utilizadas las optimizaciones de la operación colectiva alltoall, ya que alterarían los patrones de comunicación y esta prueba no tendría sentido. Todas estas pruebas han sido realizadas con un tamaño de mensaje de 2KB y la sobrecarga ha sido obtenida

mediante la comparación de los tiempos de ejecución con un tiempo de referencia mínimo, calculado a partir de la ejecución de estas mismas pruebas pero con gran cantidad de recursos destinados a buffering, y sin ningún mecanismo de control de flujo, ya que en ningún momento se producen problemas de desbordamiento de buffers, debido a su gran tamaño.

En todos los casos la sobrecarga no es adecuada (por debajo del 5%) hasta que el tamaño medio de los buffers de recepción no es mayor de 58 ranuras. Sin embargo, para los tres últimos patrones de comunicación la sobrecarga podría ser mucho menor. Para tamaños medios de buffer menores, si los buffers no usados por emisores inactivos pudieran ser asignados a emisores activos. Por ejemplo, en el caso de ocho operaciones alltoall simultáneas, un proceso dado sólo se comunica con 127 procesos, dejando 896 buffers sin usar. Esto significa que un 87.5% de los recursos de buffering están siendo desaprovechados, mientras que los procesos activos podrían aprovecharse de estos recursos.

En la siguiente sección introducimos un mecanismo de control de flujo dinámico cuyo objetivo es solventar este tipo de inconvenientes. En la sección VI mostramos una gráfica similar a la de la Figura 1 que claramente muestra como una distribución dinámica de los créditos beneficia notablemente a los procesos activos.

V. Control de flujo dinámico basado en créditos

Con el objetivo de solucionar los problemas que surgen del reparto estático del mailbox hemos evolucionado el control de flujo estático hacia un nuevo control de flujo dinámico, el cual es capaz de detectar la actividad relativa entre los distintos emisores y adecuar su cuota de créditos a dicha actividad.

A. Funcionamiento del control de flujo dinámico

El funcionamiento del control de flujo dinámico sigue los mismos principios que el estático pero añade en los receptores una función de monitorización de la actividad de los distintos emisores, de forma que cuando un receptor alcanza el umbral de retorno de créditos para un emisor dado, antes de retornarle los créditos evalua su actividad relativa respecto a la del resto de emisores y, en función de este resultado, el número de créditos retornados será mantenido, incrementado o decrementado.

Durante la inicialización, al igual que en el control de flujo estático, reservamos una región de control en el mailbox. Esta región tendrá exactamente el mismo tamaño que en el control de flujo estático, debido a que el comportamiento base del control de flujo dinámico es el mismo que el del estático. El resto del mailbox, región de datos en el control de flujo estático, será dividido en dos regiones: una *región estática*, que se corresponderá con el tamaño mínimo que puede tener la región de datos en el control de flujo estático, vista en la ecuación 2, y una *región dinámica* que estará formada por el restante espacio en el mailbox. La región estática será asignada a cada emisor al momento de inicialización de la aplicación paralela y representará la cuota de créditos mínima que un emisor podrá tener en el transcurso de toda la ejecución de la aplicación paralela. La región dinámica no será asignada a los emisores en el inicio, sino que servirá para ir incrementando la cuota de los emisores que posean una mayor actividad. Una vez iniciado el control de flujo dinámico su funcionamiento será el mismo que el visto para el estático con la principal diferencia de que en este mecanismo controlaremos la actividad de los emisores.

B. Control de la actividad

La actividad asociada a un proceso es definida exclusívamente a partir del tiempo que tarda un proceso en consumir su cuota de créditos. Debido a esto, si dos procesos consumen su cuota de créditos en la misma cantidad de tiempo, ambos tendrán la misma actividad, independientemente de la cuota de créditos que tengan asignada cada uno de ellos, por lo tanto, el mecanismo de control de flujo no podrá sustraer creditos dinámicos a uno para otorgárselos al otro.

El control de la actividad será realizado cada vez que un emisor consuma su cuota de créditos de forma íntegra. Esta situación será fácilmente detectada por los receptores, ya que se dará cada vez que alcance el umbral de retorno de créditos un número de veces igual a $n + 1$, siendo n el número de ranuras de control utilizadas. Esto es deducible de la ecuación 1. En cada evaluación de la actividad se comparará la actividad del emisor al que se le van a retornar créditos frente a la de los demás. Si existen emisores con menor actividad, se eligirá al que menor actividad posea y se le sustraerán créditos dinámicos. Después los créditos dinámicos sustraídos se añadiran a la cuota del emisor al que le vamos a retornar los créditos. Finalmente se enviará el paquete de control. De esta forma el sistema tiende a buscar un equilibrio en el que todos los procesos tarden el mismo tiempo en consumir su cuota de créditos.

VI. Resultados experimentales y análisis

El comportamiento del mecanismo de control de flujo ha sido evaluado mediante una serie de pruebas que serán mostradas a continuacion.

A. El control de flujo dinámico solventa los inconvenientes observados en el estático

Hemos repetido la misma prueba evaluada en la sección IV pero en este caso hemos usado el mecanismo de control de flujo dinámico. La Figura 2 muestra que el control de flujo dinámico sí consigue detectar el patrón de comunicación y distribuir los recursos dedicados a buffering de forma adecuada al patrón detectado. En el caso del alltoall 1x1024 no puede existir una distribución de recursos mejor que la realizada por el control de flujo estático. Por lo tanto, si comparamos los resultados obtenidos en este caso con los resultados obtenidos por el control de

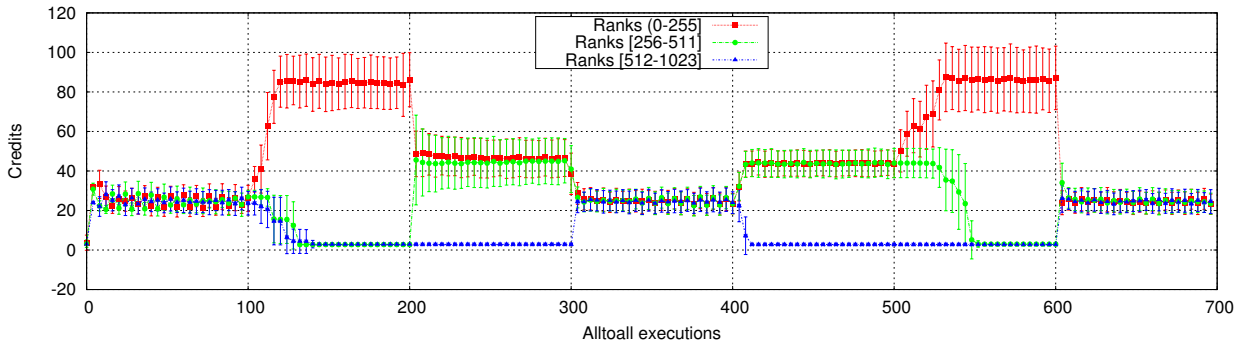

Fig. 3. Distribución de los créditos para el proceso 0

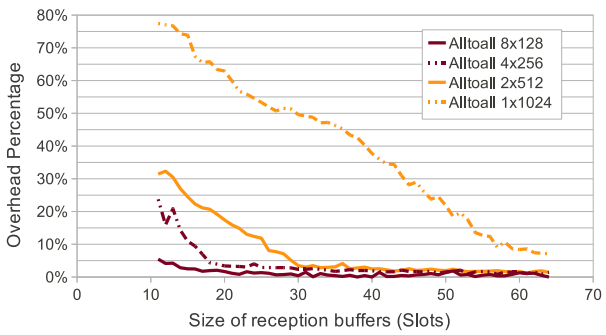

Fig. 2. Sobrecarga inducida por el control de flujo dinámico para cuatro patrones de comunicación diferentes

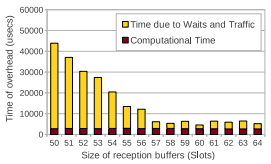

Fig. 4. Sobrecarga alltoall, caso estático

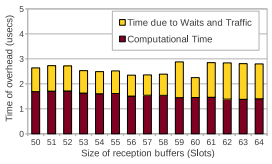

Fig. 5. Sobrecarga multi-pingpong, caso estático

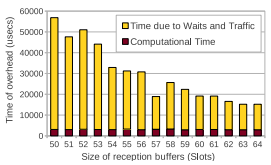

Fig. 6. Sobrecarga alltoall, caso dinámico

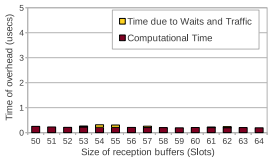

Fig. 7. Sobrecarga multi-pingpong, caso dinámico

flujo estático, mostrados en la Figura 1, vemos que la sobrecarga originada por el control de flujo dinámico es ligeramente mayor. Esto es debido a que para este caso el control de actividad realizado por el control de flujo dinámico se ve ligeramente afectado por la gran cantidad de tráfico existente en la red. En el caso 2x512 un proceso únicamente se comunica con la mitad de los restantes procesos y podemos apreciar como claramente son necesarios sólamente la mitad de recursos para que la sobrecarga se mantenga en niveles muy bajos (menores del 5%). El resto de casos pueden ser analizados de forma análoga a este último.

B. *Análisis de las componentes de la sobrecarga generada*

Básicamente la sobrecarga generada por el control de flujo es debida a dos razones:

1. El tiempo de esperas que se producen cuando un proceso emisor consume todos sus créditos y el tráfico adicional generado por el control de flujo, paquetes de control.
2. La sobrecarga debida a la computación que requiere la gestión de los créditos.

En esta prueba estudiaremos el peso de estas componentes en la sobrecarga obtenida tanto para el control de flujo estático como para el control de flujo dinámico. Las pruebas que hemos realizado muestran dos patrones de comunicación muy extremos. El primero es un alltoall, que como hemos dicho anteriormente, es el único caso en el que una repartición estática del mailbox es adecuada y el segundo es un multi-pingpong que presenta un patrón mucho más localizado. Ambos casos han sido ejecutados con un tamaño de mensaje medio de 2KB y el valor de la sobrecarga ha sido calculado a partir de un valor de referencia mínimo, calculado de igual forma que en secciones anteriores. El alltoall ha sido ejecutado con 1024 procesos, mientras que el multi-pingpong ha sido ejecutado con 32 procesos, ejecutando 16 operaciones de pingpong simultáneas. Como podemos observar en las Figuras 4 hasta la 7 para el caso del alltoall, la sobrecarga media es mayor al usar el control de flujo dinámico pero la sobrecarga computacional es la misma. Nótese que este caso se corresponde con el caso 1x1024 visto en el apartado anterior y por lo tanto este aumento en la sobrecarga es debido a la posible interferencia del tráfico extremo, presente en la red, sobre el algoritmo de control de actividad. En el caso del multi-pingpong, el mecanismo de control de flujo dinámico redistribuye los créditos en función del patrón de comunicación y como se puede apreciar la reducción de la sobrecarga debida a cada una de las componentes es muy notable.

C. *Análisis de la distribución de los créditos ante un patrón de comunicación cambiante*

En esta prueba vamos a analizar cómo el control de flujo dinámico redistribuye los créditos en función de los cambios que se producen en el patrón de comunicación. Para ello se ha realizado una aplicación paralela en la que el patrón de comunicación cambia bruscamente a lo largo de su ejecución. Esta

aplicación realiza exclusivamente llamadas a la operación colectiva alltoall pero va variando el número de procesos involucrados, de modo que el patrón de comunicación sufre cambios bruscos. Se ha realizado la prueba con 1024 procesos en la que el tamaño del mensaje intercambiado en la operación de alltoall ha sido de 2 KB, el tamaño medio de los buffers ha sido situado en 30 ranuras. Los patrones de comunicación generados han sido:

a) Extremadamente balanceado, todos los procesos se comunican con todos los demás, alltoall (0-1024)
b) Intermedio, un proceso se comunica con el 50% de los procesos restantes, alltoall (0-511)
c) Muy localizado, un proceso únicamente se comunica con el 25% de los procesos restantes, alltoall (0-255)

Han sido ejecutadas 700 llamadas a la operación colectiva alltoall. El patrón ha sido cambiado cada 100 operaciones, siendo: a,c,b,a,b,c,a. Antes de cada operación alltoall se han recogido los valores de los contadores de créditos del proceso de rango 0 hacia los restantes procesos. La Figura 3 muestra el número de créditos medio para: los procesos con rangos comprendidos entre 1 y 255 (cuadrados), los procesos con rangos comprendidos entre 256 y 511 (círculos), los procesos con rangos comprendidos entre 512 y 1023 (triángulos). Las barras de error muestran el valor de la desviación típica para estos valores medios. Como puede observarse, la distribución de los créditos se ajusta perfectamente al patrón de comunicación en cada caso.

D. *Evaluación mediante Intel MPI Benchmarck Suite*

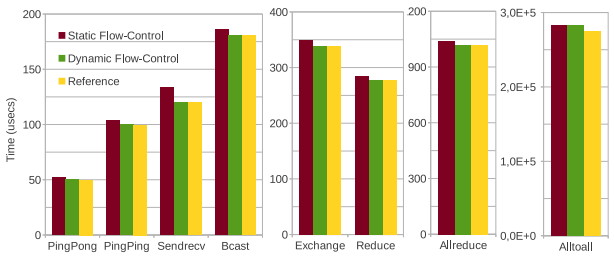

Fig. 8. Resultados obtenidos para diferentes pruebas contenidas en IMB

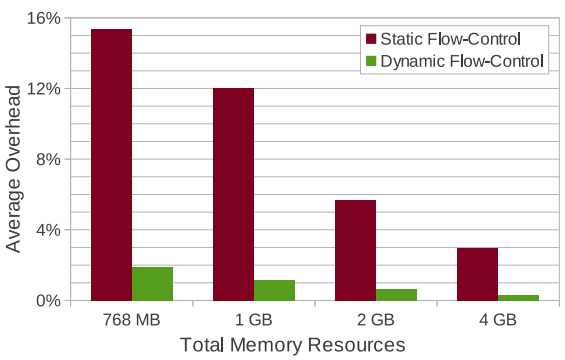

Fig. 9. Sobrecarga media generada por los mecanismos de control de flujo en función de los recursos destinados a buffering

Finalmente realizamos una última prueba en la que se han elegido diferentes tests de la suite de benchmarks Intel MPI [7] y se han comparado los tiempos de ejecución del control de flujo estático y dinámico frente a un tiempo de referencia mínimo. La Figura 8 muestra los resultados obtenidos para las ejecuciones de los distintos tests con 1024 procesos, un tamaño de mensaje de 2KB y para los casos estático y dinámico con un tamaño medio de buffers de 64 slots, lo que supone un total de 4GB de memoria dedicados a buffering en todo el cluster. Con el fin de observar la eficiencia de ambos mecanismos de control de flujo hemos ido reduciendo los recursos dedicados a buffering. La Figura 9 muestra la sobrecarga media para los tests anteriores, generado por los mecanismos de control de flujo estático y dinámico. Podemos observar que la eficiencia del control de flujo dinámico es muy alta, ya que en el caso en el que reducimos más los recursos de buffering la sobrecarga media obtenida es menor del 2% en el caso dinámico, mientras que en el caso estático es superior a un 15%.

VII. Conclusiones

En este artículo se ha presentado el desarrollo de un nuevo mecanismo de control de flujo, el cual ha sido el resultado de la evolución de un primer mecanismo de control de flujo estático basado en créditos a un mecanismo de control de flujo dinámico, capaz de detectar el patrón de comunicación subyacente de una aplicación paralela y distribuir los recursos de buffering de forma acorde a este patrón en tiempo de ejecución. El análisis de este nuevo mecanismo de control de flujo sobre la implementación MPI de EXTOLL muestra un gran nivel de eficiencia frente a la clásica repartición de recursos de buffering estática.

Agradecimientos

Este trabajo fue financiado por Spanish MICINN, Plan E funds, under Grant TIN2009-14475-C04-01.

Referencias

[1] EXTOLL homepage, *http://www.extoll.de/*
[2] InfiniBand homepage, *http://www.infinibandta.org/*
[3] MPI homepage, *http://www.mpi-forum.org/*
[4] OpenMPI homepage, *http://www.open-mpi.org/*
[5] D. Bonachea and J. Jeong, *GASNet: A Portable High-Performance Communication Layer for Global Address-Space Languages*, CS258 Parallel Computer Architecture Project, Spring 2002
[6] K. Kuyat, F. Silla, H. Fröning, and J. Duato, *The EXTOLL Conduit for the GASNet networking layer*, JP2012
[7] Intel MPI Benchmarks homepage, *http://software.intel.com/en-us/articles/intel-mpi-benchmarks/*
[8] J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G.E. Fagg, E. Gabriel, and J.J. Dongarra, *Performance Analysis of MPI Collective Operations*, IPDPS 2005
[9] U. Brüning, and H. Fröning, *HTX-Board v1.3 User Manual*, Computer Architecture Group, University of Heidelberg, Germany.
[10] Advanced Micro Devices homepage, *http://www.amd.com/*
[11] H. Litz, et al. *A novel communication engine for ultra-low latency message transfers.*, ICPP 2008
[12] M. Nüssle, et al., *A resource optimized remote-memory-access architecture for low-latency communication.*, ICPP 2009
[13] S. S. Mukherjee, P. Bannon, S. Lang, A. Spink, D. Webb, *The Alpha 21364 Network Architecture*, Journal IEEE Micro, Volume 22 Issue 1, January 2002, Pages 26 - 35