Migración Dinámica de Procesos: El Problema de la Integridad de Mensajes

E. Heymann F. Tinetti E. Luque

Universidad Autónoma de Barcelona
Departamento de Informática
Unidad de Arquitectura de Ordenadores
08193 - Bellaterra, Barcelona, España
e-mail: iinfd@cc.uab.es

Resumen [1]

La gestión de los recursos procesador y red de interconexión tiene un gran impacto en las prestaciones de los computadores paralelos de memoria distribuida. La migración dinámica de procesos permite hacer el balance de carga y de comunicaciones en tiempo de ejecución. Uno de los problemas que implica la migración dinámica consiste en la gestión de las comunicaciones que involucran al proceso que migra, el cual debe seguir recibiendo todos los mensajes que se le envían independientemente de su ubicación en la red. Como primera fase de estudio de este problema, al cual hemos llamado Problema de Integridad de Mensajes, se han desarrollado seis algoritmos que garantizan la recepción de los mensaje en presencia de migración dinámica de procesos: Esquema Centralizado, Enviar Primero a la Dirección Hogar, Buzones para Migración, Sígueme, Rechazo de Mensajes y Protocolo Completo. Estos algoritmos han sido estudiados en primer lugar mediante simulación secuencial, y posteriormente mediante implementación en una máquina paralela, bajo la cual se han ejecutado los algoritmos para diferentes patrones de procesos de usuario en presencia de migración dinámica. Los parámetros de prestaciones considerados para evaluar y comparar los algoritmos han sido los siguientes: la latencia de los mensajes; la carga extra añadida sobre la red, que viene dada por las retransmisiones y los mensajes de control que cada algoritmo genera; el tiempo de reacción, que representa el tiempo transcurrido desde que un proceso es elegido para migrar hasta que puede ser migrado preservando la recepción de sus mensajes; la escalabilidad; y la influencia del algoritmo sobre el patrón de comunicaciones. Los resultados obtenidos nos han dado información preliminar sobre el comportamiento de los algoritmos, y nos han permitido realizar una primera evaluación comparativa entre los mismos.

1. Introducción

El paralelismo permite obtener altas prestaciones replicando el hardware del computador. Un computador paralelo de memoria distribuía (CPMD) está organizado como un conjunto de nodos que se comunican mediante una red de interconexión. Cada uno de estos nodos está compuesto por un procesador y memoria. La comunicación entre procesos puede llevarse a cabo mediante paso explícito de mensajes, o mediante memoria compartida virtual. En los CPMD se combinan altas prestaciones con escalabilidad. Sobre estas máquinas, se ha considerado un modelo de programación basado en el paso de mensajes. La unidad de paralelismo es el *proceso*, el cual es una entidad lógica que ejecuta código secuencial. Los *canales* permiten la comunicación y sincronización entre procesos.

Las prestaciones de los CPMD pueden mejorarse con una utilización balanceada de sus recursos. Esto significa balancear la carga de cómputo (balanceo de carga) y balancear la utilización de la red (balanceo de comunicaciones). Es difícil lograr estas metas de manera estática, ya que normalmente no se conoce a priori el comportamiento de las aplicaciones paralelas[1]. Cuando un procesador tiene asignados algunos procesos, con necesidades elevadas de cómputo, sería conveniente separar dichos procesos, esto es, colocarlos en diferentes procesadores, balanceado de esta manera la carga de la aplicación paralela. De la misma manera, si un procesador tiene asignados procesos con necesidades de comunicación mayores que el ancho de banda en dicho nodo, se produciría una congestión (*hot spot*). En estos casos sería también de utilidad separar dichos procesos, logrando así un balanceo de las comunicaciones. Por lo tanto, se espera obtener mejores prestaciones mediante la Migración Dinámica de Procesos. El soporte para la migración de procesos permite cambiar la asignación de procesos a procesadores de manera dinámica, esto es, en tiempo de ejecución [2].

[1] Este trabajo ha sido soportado por la Comisión Interministerial de Ciencia y Tecnología (CICYT) española, bajo número de contrato TIC 95/0868, y parcialmente patrocinado por el Programa Copernicus de la UE bajo números de contrato CIPA-C1-93-0251 y CIPA-CP-93-5383.

De la misma forma que los mecanismos de traslación de direcciones y el movimiento automático de datos a través de la jerarquía de memoria (memoria virtual/ memoria cache) se han impuesto como características fundamentales de cualquier computador secuencial, pensamos que también será necesario incluir al mecanismo de migración de procesos en el hardware, mejorando de esta forma las prestaciones de los computadores paralelos.

La migración dinámica de procesos se basa en las siguientes tareas:

- Evaluar la carga de los procesadores y de la red, y determinar como utilizar esta información para decidir que proceso debe migrar, a dónde y cuando migrará (Políticas de Balanceo Dinámico de Carga);
- La migración física en sí, esto es, cómo migrar. Esto incluye el desalojo de un proceso en el procesador fuente, y su reubicación en el procesador destino (Mecanismos de Migración de Procesos);
- Gestionar las comunicaciones que involucran al proceso que migra, el cual debe continuar recibiendo mensajes independientemente de su ubicación en la red (Gestión de la Integridad de los Mensajes).

Nuestro trabajo se concentra en el último punto[3], al cual hemos llamado *Problema de la Integridad de Mensajes* (PIM). Dada una asignación inicial de procesos a procesadores, en la cual cada proceso S1, S2, ... , Sk conectado con el proceso R, conoce la ubicación de R en la red, esto es, la identificación del procesador Pi en el cual R se ejecuta, en una máquina basada en el paso de mensajes, el PIM consiste en mantener estas conexiones una vez que el procesos R migra desde del procesador Pi al procesador Pj, como se muestra en la figura 1.

Otra situación en la que aparece el PIM es la siguiente: Existen muchas copias del proceso *p*, cada una de ellas en un procesador diferente, pero solo una de ellas está activa en un momento dado. En este caso un algoritmo de integridad de mensajes es necesario para garantizar que un mensaje dirigido a *p* llegará a la ubicación donde se encuentra la copia activa de *p*. Es interesante mencionar la similitud entre el PIM y el problema de acceso a memoria cache en una máquina paralela de memoria compartida [4]. La ubicación original del proceso que migra sería análoga a la dirección de un dato compartido que se encuentra en una memoria cache perteneciente a un procesador en particular; y enviar un mensaje a un proceso sería análogo a leer o escribir datos compartidos.

El PIM implica considerar que acciones son tomadas por (1) el proceso que migra, R; y por (2) los procesos que le envían mensaje a R. Para tratar el PIM se han desarrollado seis algoritmos que presentan diferentes características: *Esquema Centralizado*, *Enviar Primero a la Dirección Hogar*, *Buzones para Migración*, *Sígueme*, *Rechazo de Mensajes* y *Protocolo Completo*. Con el objeto de poder compararlos entre sí, éstos algoritmos han sido estudiados por simulación secuencial y han sido implementados en una máquina paralela, sobre la cual se han ejecutado para diferentes patrones de procesos usuario cuando la migración dinámica de procesos es soportada. Los parámetros de prestaciones que han sido considerados para evaluar y comparar los algoritmos propuestos son: la latencia de los mensajes, la carga extra producida, el tiempo de reacción, la escalabilidad y como los algoritmos modifican el patrón de comunicaciones sobre la red.

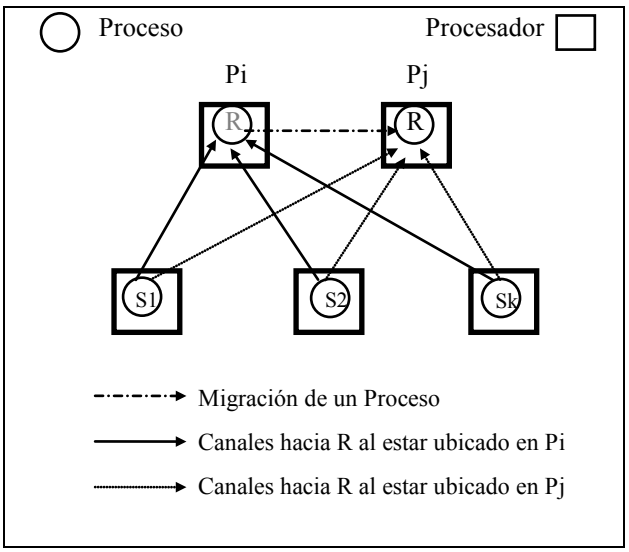

Figura 1. El Problema de la Integridad de Mensajes

Por razones de simplicidad, pero sin pérdida de generalidad, sólo se considera creación estática de procesos de usuario y de canales de comunicación.

El resto del artículo esta organizado de la siguiente manera: Los algoritmos que resuelven el problema de la integridad de mensajes están descritos en la sección 2; en la sección 3 se encuentran los parámetros de evaluación y la experimentación realizada con los algoritmos; cuyos resultados permiten conocer el comportamiento de cada algoritmo. Estos resultados se encuentran en la sección 4; la sección 5 contiene el análisis de los resultados anteriores; y en la sección 6 se exponen las conclusiones y las líneas que este trabajo deja abiertas.

2. Descripción de los Algoritmos

Para describir los algoritmos que resuelven el problema de la integridad de mensajes, se ha utilizado la siguiente convención: sea R un proceso que recibe mensajes y que migra del procesador Pi al procesador Pj, y sean S1, S2, ... , Sk los procesos conectados a R a través de canales estáticos, y que le envían mensajes.

La carga extra de comunicación que cada algoritmo añade esta compuesta por: (1) Los *mensajes de retransmisión*; se producen cuando un mensaje llega a un destino erróneo y debe ser retransmitido para alcanzar su destino correcto, debido al hecho de que el proceso destino R había migrado; y (2) Los *mensaje de control* que vienen dados por los mensajes necesarios para implementar el algoritmo de integridad de mensajes.

· **Esquema Centralizado:** Existe un Proceso Servidor Central (SC) que proporciona las direcciones físicas (identificación de procesadores) de los procesos destino, a través de su tabla de asignación de procesos a procesadores. Este SC debe ser notificado de cualquier migración, esto es, cuando un proceso migra debe enviarle su nueva ubicación al SC. Cuando un proceso Si desea enviar un mensaje al proceso R, debe realizar los siguientes pasos:

(1) Preguntar al Servidor Central la ubicación de R.

(2) Esperar la respuesta del SC -por ejemplo Pi-. Esta es la última dirección conocida de R.

(3) Enviar el mensaje deseado a Pi.

Es posible que un mensaje sea enviado a una dirección errónea. Esta situación puede producirse cuando la respuesta del servidor a Si es enviada justo antes de que el Servidor Central reciba la notificación de la migración de R, y por lo tanto reciba su nueva ubicación. En este caso se repiten los pasos 1-3; y el proceso Si recibe una señal NACK.

Cada operación de envío de mensaje implica en la mayoría de los casos dos mensajes de control, uno para preguntarle al SC la dirección del proceso destino R, y otro para recibir su respuesta. Si R ha migrado pero el SC no ha sido aun notificado, cualquier mensaje destinado a R llegará a un destino erróneo, caso en el cual se realizará una retransmisión y se generarán mas mensajes de control (NACK). Cada operación de migración genera un mensaje de control para notificar al SC la nueva ubicación del proceso que migra.

· **Enviar Primero a la Dirección Hogar:** El *procesador hogar* es el procesador inicial dónde cada proceso es asignado. Suponiendo que el proceso R ha sido inicialmente asignado al procesador Pi (procesador hogar de R), cualquier mensaje destinado a R será enviado al procesador Pi, sin importar sí R está aún allí. Una vez en Pi, sí R ha migrado, el mensaje es retransmitido a la nueva ubicación de R, digamos Pj. Esto implica que el procesador hogar debe conocer siempre la ubicación de los procesos que le hayan sido asignados inicialmente. Cuando un proceso migra, debe enviar su nueva ubicación a su procesador hogar; de esta manera el procesador hogar será capaz de retransmitirle los mensajes a R. Es posible que algunos mensajes destinados a R hayan sido enviados a Pj antes que el procesador hogar de R haya sido notificado de la nueva ubicación de R. Estos mensajes no encontrarán a R en Pj, con lo cual se generará una señal NACK que será enviada al procesador hogar. Una vez que éste reciba la nueva ubicación de R, le retransmitirá el mensaje.

Cada migración produce un mensaje de control destinado al procesador hogar de R, a fin de notificarle su nueva ubicación. Una vez que R ha migrado, todos los mensajes que le sean enviados irán primero a su procesador hogar y luego le serán retransmitidos. Se producirán retransmisiones adicionales cuando R migre, su procesador hogar no haya sido aún notificado, y llegue un mensaje destinado a R.

· **Buzones para Migración:** La idea bajo este esquema consiste en modificar la semántica de la primitiva *recibir*. Cuando el proceso R ejecuta un *receive*, se envía un mensaje de solicitud a una dirección predefinida, llamada *buzón para migración*. Cualquier proceso Si que desee comunicarse con el proceso R, enviará el mensaje al buzón de R, independientemente de la ubicación de R, es decir, de si R a migrado o no. Esto implica que no se toma ninguna acción cuando un proceso migra, esto es, que nadie ha de ser notificado de la migración de R. Este esquema comporta una

restricción: sí un proceso debe migrar, debe esperar a recibir todos los mensajes que haya solicitado a su buzón, antes de que la migración pueda efectuarse.

El buzón de un proceso R está ubicado en el mismo procesador dónde R fue asignado inicialmente, digamos Pi. Por lo tanto cualquier mensaje destinado a R será inicialmente enviado al procesador Pi. Cuando R y su buzón están en procesadores diferentes, esto es, una vez que R ha migrado, cada operación de recepción producirá un mensaje de control y una retransmisión.

· **Sígueme:** Cuando el proceso R migra deja su nueva ubicación (dirección de migración) en el procesador que abandona. De esta manera cada proceso construye un camino que deben seguir los mensajes destinados a él. Cuando un proceso Si desea enviarle un mensaje a R, le envía el mensaje al procesador dónde R estaba inicialmente asignado, digamos Pi. Si R ha migrado, la ubicación de R dejada en Pi es utilizada para seguir al proceso R. Este paso es realizado tantas veces como sea necesario para alcanzar a R.

Por un lado este algoritmo no genera mensajes de control, pero en cambio, una vez que un mensaje es inyectado en la red, será retransmitido hasta que alcance a R, esto es, tantas veces como R haya migrado, a menos que R migre a un procesador en el cual ya haya estado, caso en el cual el camino se reduce.

· **Rechazo de Mensajes:** Cuando un proceso migra del procesador Pi al procesador Pj, su nueva ubicación es almacenada en el procesador que abandona. Cuando un proceso Si le envía un mensaje a R utiliza la última información que posea sobre la ubicación de R, que representa la ubicación de R cuando Si le envió su último mensaje. Si un procesador recibe un mensaje para R, y R ya no se encuentra en dicho procesador, se produce un rechazo del mensaje, para lo cual envía una señal NACK al procesador en el cual se encuentra Si, junto con la ubicación que el conoce de R. El procesador de Si actualiza su ubicación de R, utilizando la ubicación recién recibida y retransmite el mensaje. Estos pasos se repiten hasta que el mensaje alcance a R.

Cada vez que se rechaza un mensaje se genera un mensaje de control (NACK) que contiene la ultima ubicación conocida de R, y una retransmisión del mensaje a la ubicación recibida..

· **Protocolo Completo:** El proceso R no podrá migrar hasta que (a) todos los procesos S1, S2, ... , Sk conectados con él conozcan su nueva ubicación, y (b) R reciba todos los mensajes que le hayan sido enviados.

Cuando el proceso R migra, realiza un protocolo de intercambio de información con todos los procesos S1, S2, ... , Sk. Este protocolo está compuesto de los siguientes pasos:

(1) R envía una seña a todos los procesos S1, S2, ... , Sk a fin de que dejen de enviar mensajes a R;

(2) estos procesos le contestan a R, informándole el número de mensajes que ellos ya han inyectado en la red;

(3) R espera la llegada de todos los mensajes pendientes;

(4) cuando R ha recibido todos los mensajes pendientes, migra a Pj;

(5) R avisa a los procesos S1,S2,... , Sk su nueva ubicación y los autoriza a que le continúen enviando mensajes.

Este algoritmo no genera ninguna retransmisión, pero cada vez que un proceso R migra, se generan 3 * k mensajes de control, representando k el número de procesos conectados a R.

Una variación de este algoritmo permite que R migre después de paso 2. En este caso los mensaje pendientes serán retransmitidos de Pi a Pj.

3. Parámetros de Evaluación y Experimentación

Los algoritmos propuestos han sido evaluados y comparados en base a los siguientes parámetros de prestaciones:

· *Latencia*: El tiempo promedio para enviar un mensaje de un proceso a otro, esto es, el tiempo que transcurre desde que el mensaje deja el proceso origen y alcanza al proceso destino.

· *Carga sobre la red:* La carga extra que cada algoritmo añade sobre la red, y viene dada por el número promedio de retransmisiones por mensaje, más el número promedio de mensajes de control. Este parámetro indica el ancho de banda que consume un algoritmo.

· *Tiempo de reacción:* Representa el tiempo transcurrido desde que un proceso es seleccionado para migrar, hasta que dicho proceso puede ser migrado conservando la integridad en la recepción de sus mensajes.

· *Escalabilidad:* Cómo es el comportamiento del algoritmo cuando se añaden procesos y procesadores.

· Como el algoritmo modifica el *patrón de comunicaciones*. Hacer énfasis en el patrón de comunicaciones es necesario para evaluar si el algoritmo es capaz de realizar un balanceo de las comunicaciones.

Para evaluar estos parámetros de prestaciones se ha realizado la siguiente experimentación, que incluye simulación e implementación:

· Modelado y simulación secuencial de los algoritmos propuestos. Esta simulación tuvo por objeto el poder analizar el comportamiento intrínseco de cada algoritmo, sin la influencia de elementos externos como la carga sobre la red de comunicaciones o el patrón de envío de mensajes que posea la aplicación.

· Implementación de los diferentes algoritmos que resuelven el MIP en una máquina paralela. En esta implementación la aplicación, es decir los procesos de usuario, ha sido simulada mediante un programa sintético que simula cómputo, y envía y recibe mensajes reales. La migración de procesos también se simula, pero al igual que los mensajes de comunicación, genera tráfico real sobre la red. El programa sintético permite realizar ejecuciones modelando diversas situaciones. La latencia de los mensajes se ve afectada por la carga sobre la red de comunicaciones. Los mensajes entre procesos, los mensajes de control y retransmisiones y los procesos de usuario que migran son transmitidos utilizando la red de comunicaciones, y constituyen carga real.

Para cada uno de los seis algoritmos propuestos se han realizado simulaciones y ejecuciones variando diversos parámetros. Se consideraron 8, 32 y 64 procesadores para las simulaciones; y 8, 16 y 32 procesadores para las ejecuciones. Este parámetro ha permitido estudiar la escalabilidad de los algoritmos. La relación procesos/procesadores contempló los valores de 5 y 20 procesos por procesador. Este parámetro no ha sido significativo en los resultados. Cada vez que se genera una operación de envío de mensajes, se produce una migración con probabilidad p, que representa el porcentaje de migración. Diversos porcentajes de migración fueron estudiados: 0%, 1%, 5% y 10%. El porcentaje de migración de 0% se utilizó como referencia, y representa el caso en el cual existe un mecanismo que soporte la migración dinámica de procesos, pero ningún proceso migra. Tanto las simulaciones como las ejecuciones finalizan cuando se hayan generado un promedio de 150 mensajes por proceso.

Es importante señalar que todos los procesos están conectados entre sí, esto es, el proceso R puede recibir mensajes desde cualquier otro proceso; y que al añadir procesos se escala la conectividad. Este hecho penaliza el algoritmo del Protocolo Completo, quien a tiempo de migración realiza comunicaciones con todos los procesos conectados al proceso que migra, sin contar con un mecanismo de *multicast*. Para evaluar el efecto del número de conexiones que tienen los procesos, todos los algoritmos han sido simulados con los siguientes grafos de procesos: (1) grafo completo, en el cual todos los procesos están conectados entre sí; (2) pipe bidireccional, en el cual cada proceso está conectado con otros dos, y el número de conexiones presenta un crecimiento lineal al escalar ; y (3) una topología de hipercubo, donde en un hipercubo de 2^n procesos, cada proceso está conectado con n procesos, y el número de conexiones tiene un crecimiento logarítmico al escalar.

Para cada algoritmo se realizaron 24 simulaciones y ejecuciones. Adicionalmente todos los algoritmos fueron simulados para las diferentes topologías anteriormente descritas.

3.1 La Simulación Secuencial

Los simuladores secuenciales han sido implementados utilizando un modelo de simulación de eventos discretos. La latencia considerada es la mínima sobre una red de procesadores, ya que no se considera en esta aproximación el efecto sobre la latencia producido por la carga de la red, añadida por los algoritmos. Como la latencia y la carga sobre la red se manejan de manera independiente, a pesar de que la sobrecarga producida por los algoritmos descritos incrementa la carga, no afecta la latencia.

El modelo de la arquitectura considerado en la simulación secuencial soporta la comunicación entre cada par de procesos, independientemente de su ubicación en la red de procesadores [5], [6]. Se considera que cada procesador está conectado con todos los demás. A fin de obtener una latencia uniforme, se realiza un encaminamiento aleatorio en dos pasos. El tiempo de comunicación entre cada par de procesos es una variación respecto a una función exponencial sobre un valor uniforme que representa dicho encaminamiento de dos pasos. El primer paso es un encaminamiento aleatorio, y en el segundo paso el mensaje es encaminado a su procesador destino. El tiempo promedio de comunicación viene dado por la distancia promedio entre cada par de procesadores en una topología hipercubo, la cual tiene un crecimiento logarítmico; este valor es duplicado a fin de simular en encaminamiento en dos pasos. De esta forma, el tiempo de comunicación viene dado por una función exponencial alrededor de los siguientes valores: 2.8 en un hipercubo de 8

procesadores, 5.0 en un hipercubo de 32 procesadores y 6.0 en un hipercubo de 64 procesadores. Así pues, la latencia para un mensaje en particular es proporcional al número de saltos (hops) que necesita un mensaje para llegar a su destino.

Si un mensaje llega a un servidor, por ejemplo un mensaje de solicitud se direcciones al servidor global (algoritmo del *Esquema Centralizado*), o una petición de mensajes al buzón (algoritmo *Buzones para Migracion*) , y el servidor está ocupado, el mensaje se coloca en la cola del servidor, y será atendido después que sean atendidos todos los mensajes que estaban delante de él en dicha cola.

3.2 La Implementación Paralela

La implementación paralela puede ser descrita en términos de los procesos de la fig. 2, que pueden ser modificados para cada política propuesta. El modelo sintético de la aplicación está compuesto por procesos de usuario que son asignados a los procesadores donde se simula su ejecución.

El proceso *usrsim* (proceso para simular procesos de usuario) simula el comportamiento de los procesos de usuario en posesión del procesador (CPU), como sucedería en los sistemas de multiprocesamiento. Recibe desde el proceso *coord* la descripción de un proceso de usuario (básicamente la probabilidad de envío de un mensaje y el valor de cómputo), y simula al proceso de usuario. Si el proceso de usuario no se comunica en el tiempo de procesador que le corresponde, el proceso *usrsim* consume el tiempo que le corresponde y requiere del proceso *coord* otro proceso de usuario para simular. Si el proceso de usuario se comunica, el proceso *usrsim* requiere del proceso *consrv* el otro proceso de usuario con el que se producirá la comunicación, y envía un requerimiento de mensaje al proceso *msgman*.

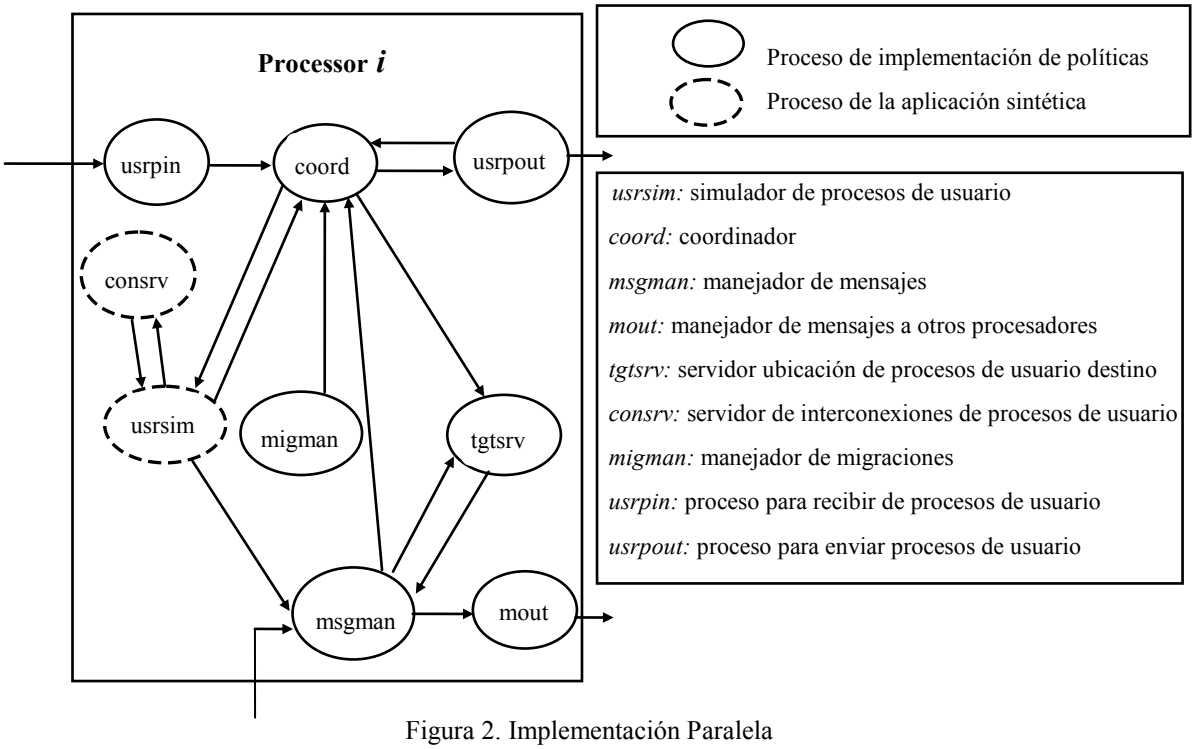

Figura 2. Implementación Paralela

El proceso *coord* (proceso coordinador) mantiene a los procesos de usuario en ejecución y además los migra. Los procesos de usuario se envían para simular su ejecución al proceso *usrsim*. Cuando recibe un requerimiento de migración hacia otro procesador desde el proceso *migman* elige aleatoriamente un proceso de usuario y lo envía a un procesador también elegido aleatoriamente utilizando el proceso *usrpout*. El proceso *tgtsrv* es notificado de cada migración.

El proceso *msgman* (proceso dedicado a manejar mensajes) se encarga de los requerimientos de mensajes. Estos requerimientos llegan desde otros procesadores o desde el proceso *usrsim* ejecutándose en el mismo procesador. Para cada requerimento, el proceso *msgman* pide al proceso *tgtsrv* la ubicación (procesador) del proceso de usuario destino del mensaje. Si el proceso de usuario destino está ejecutándose localmente el mensaje es consumido, sino, el requerimiento se maneja de acuerdo a la política que se use para resolver el PIM. Cuando un requerimiento tiene que ser enviado a otro procesador, el proceso *msgman* lo envía al proceso *mout*.

El proceso *mout* (proceso dedicado a manejar mensajes hacia otros procesadores) se dedica a la distribución de los requerimientos de mensajes hacia los demás procesadores de la red.

El proceso *tgtsrv* (proceso servidor de ubicación de procesos de usuario destino) responde a los requerimientos sobre la ubicación de los procesos de usuario. Mantiene actualizada como mínimo la información sobre los procesos de usuario que están asignados al procesador local. Recibe información de los procesos que llegan o salen del procesador (por las migraciones) desde el proceso *coord*.

El proceso *consrv* (proceso servidor de interconexiones de los procesos de usuario) conoce el grafo de interconexión entre los procesos de usuario. Una vez que un proceso de usuario ha decidido enviar un mensaje, el proceso *usrsim* requiere del proceso *consrv* el proceso de usuario destino del mensaje. Dado un proceso de usuario que enviará un mensaje, el proceso *consrv* selecciona al azar el proceso de usuario destino.

El proceso *migman* (proceso para manejar las migraciones) solamente genera requerimientos de migraciones hacia otros procesadores para el proceso *coord*. El proceso *migman* implementa la política de migración de los procesos de usuario.

Los procesos *usrpin* y *usrpout* (procesos dedicados a migrar procesos de usuario desde y hacia otros procesadores) manejan la migración física de los procesos de usuario.

El proceso *mout* en cada procesador debe comunicarse con los procesos *msgman* de cada uno de los demás procesadores, y el proceso *usrpout* debe comunicarse con todos los procesos *usrpin* en los otros procesadores. De esta forma se definen dos circuitos de datos separados: uno para datos de mensajes de usuario (*mout* - msagman), y otro para los datos de la migración de procesos de usuario (*usrpout* - *usrpin*). Si se producen cambios en uno de los circuitos no se afectará al otro circuito. Esta estructura general de procesos se requiere en cada procesador y es inicializada por procesos cargadores de la aplicación en la red de procesadores.

La simulación de las aplicaciones definidas por el usuario pueden ser definidas de acuerdo a (1) cantidad de procesos de usuario; (2) comportamiento de cada uno de los procesos de usuario en términos de cómputo y comunicaciones; (3) patrón de interconexión entre procesos de usuario; (4) probabilidad de migración de los procesos de usuario; y (5) política de migración de los procesos de usuario. Como las políticas son implementadas en una red de procesadores real, y por lo tanto no es posible en un procesador conocer el estado global del sistema (no hay un reloj común para todos los procesadores, por ejemplo), se hace necesario realizar algunas medidas locales para calcular la latencia de los mensajes. Estas medidas se realizan en cada procesador y se utilizan para calcular la latencia: (1) *lncola*: Longitud de la cola de requerimientos de mensajes cuando éstos llega al proceso *msgman*; (2) *lathop*: Latencia de salto (tiempo necesario para enviar un requerimiento de mensaje desde un procesador a otro); y (3) *hops*: Cantidad de saltos que realiza un mensaje hasta llegar al destino. Estas tres medidas se tomas localmente sin conocer el estado general de la computadora paralela. Teniendo los promedios de las medidas *lncola*, *lathop* y *hops*, la latencia promedio de los mensajes se calcula como: *lncola x lathop x hops*.

4. Resultados

En esta sección se muestran los resultados mas significativos de las simulaciones y ejecuciones realizadas. Las figuras 3a, 3b y 3c muestran los resultados de la simulación secuencial para todos los algoritmos, considerando la latencia, las retransmisiones y los mensajes de control para 8 y 64 procesadores. El eje X representa el porcentaje de migración. En la gráfica de la latencia, el eje Y representa el tiempo, en la gráfica de las retransmisiones el eje Y representa el promedio de retransmisiones por proceso, normalizado respecto al número de mensajes generado por cada proceso, esto es, 150; y en la gráfica de mensajes de control el eje Y representa el número promedio de mensajes de control por proceso, también normalizado respecto al número promedio de mensajes que genera cada proceso. Dado que las conexiones entre los procesos de usuario afecta el número de mensajes de control generados en el algoritmo del Protocolo Completo, la figura 3d muestra el número de mensajes de control producidos por esta política para procesos de usuario con topología de conexiones de hipercubo, *pipe* y grafo completo. Los resultados completos de las simulaciones secuenciales se encuentran en [7].

Las figuras 4a, 4b y 4c muestran los resultados obtenidos en las ejecuciones de la implementación paralela para 8 y 32 procesadores (límite de la máquina paralela). Las medidas y los ejes están organizados como se explicó anteriormente para las simulaciones.

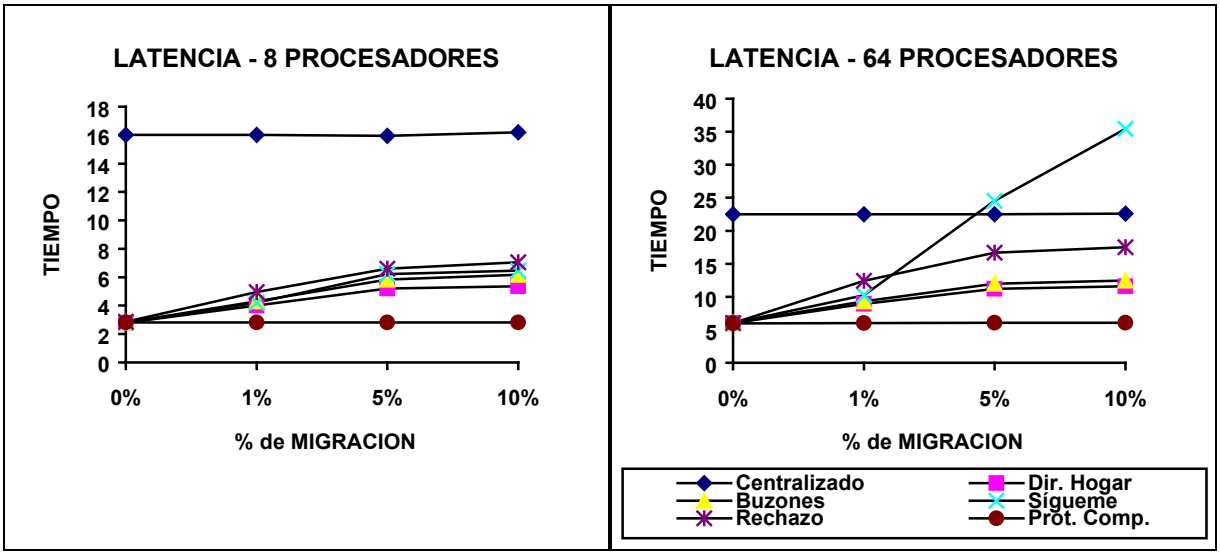

Figura 3a: Simulación Secuencial. Latencia.

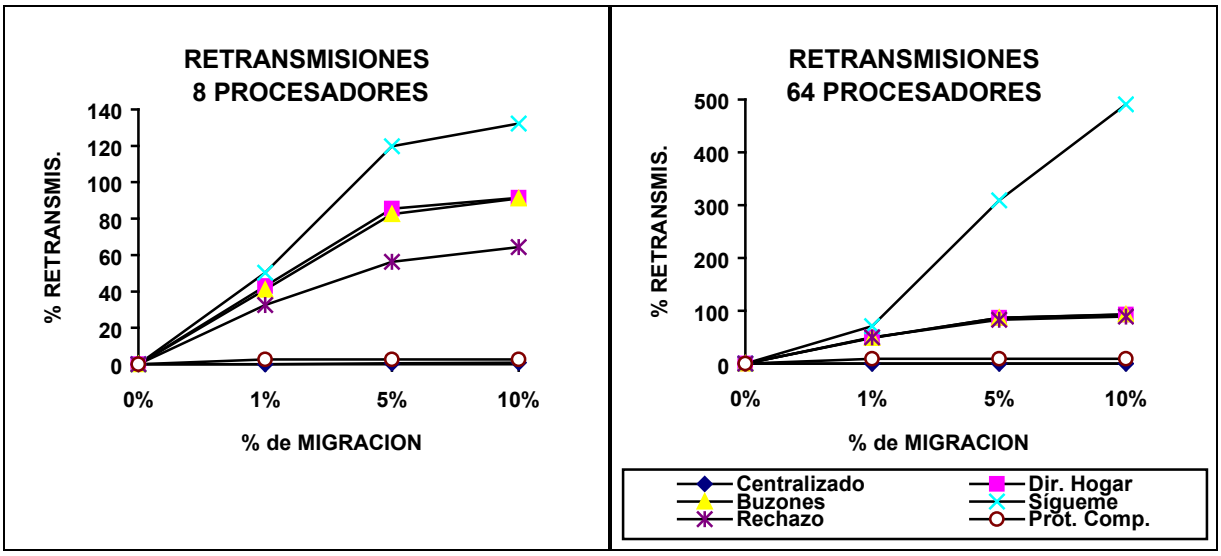

Figura 3b. Simulación Secuencial: Retransmisiones.

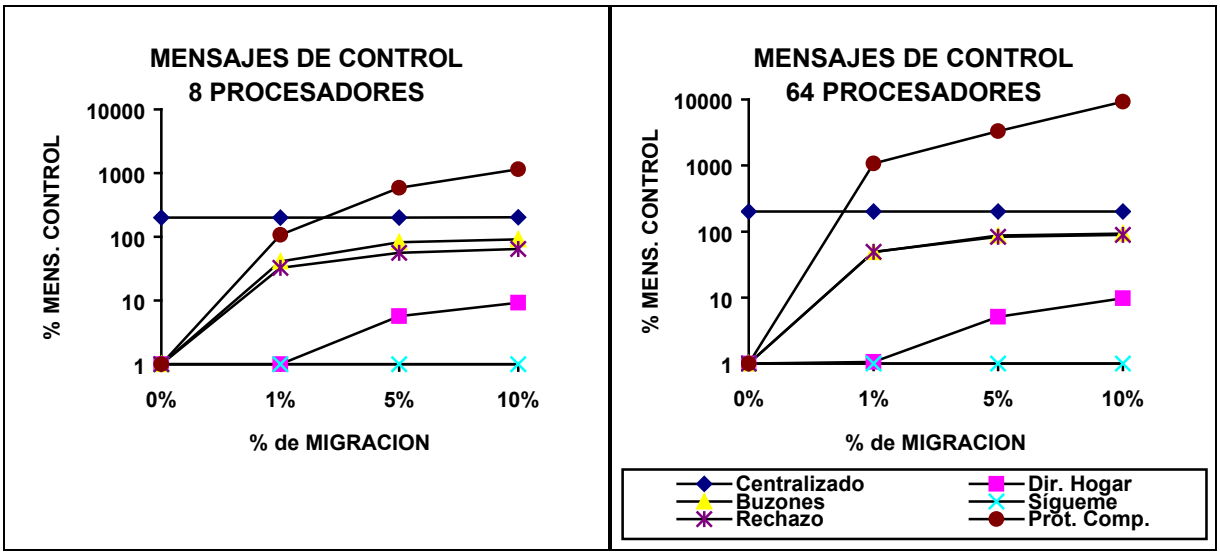

Figura 3c. Simulación Secuencial: Mensajes de Control.

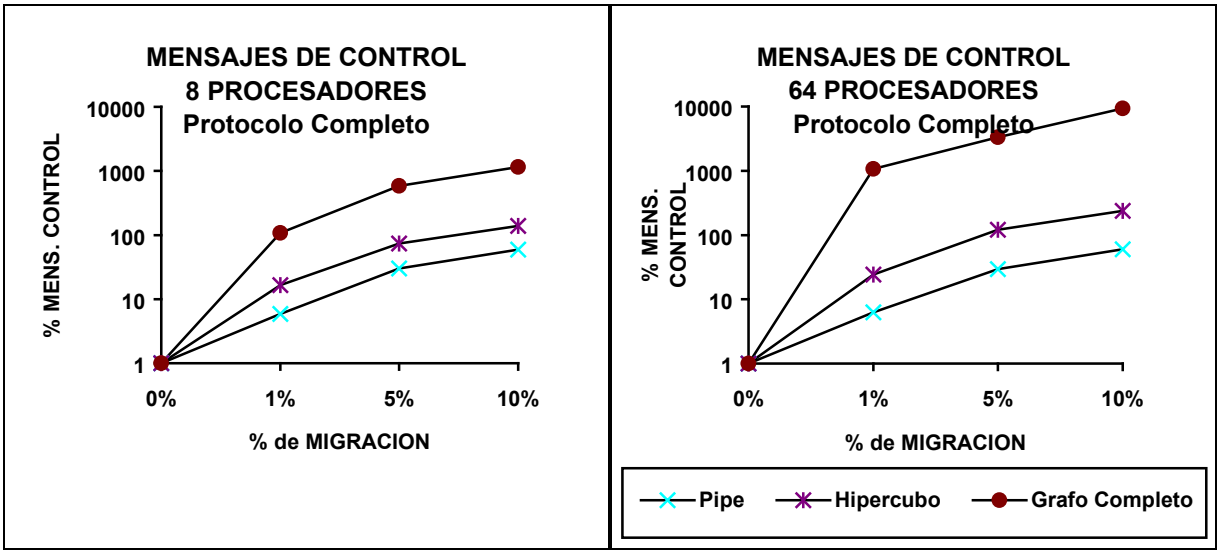

Figura 3d. Simulación Secuencial: Protocolo Completo. Mensajes de Control.

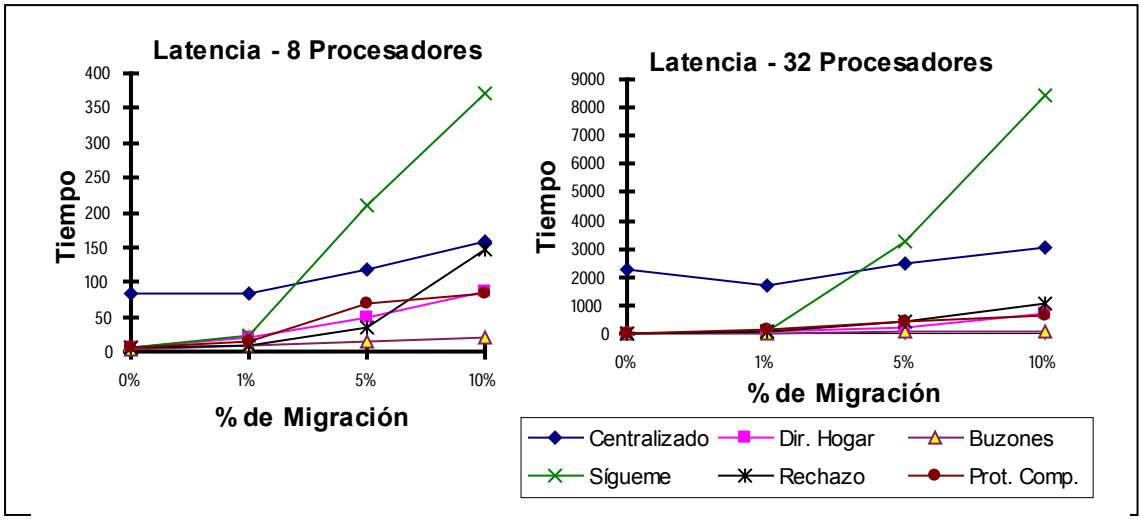

Figura 4a. Implementación Paralela: Latencia

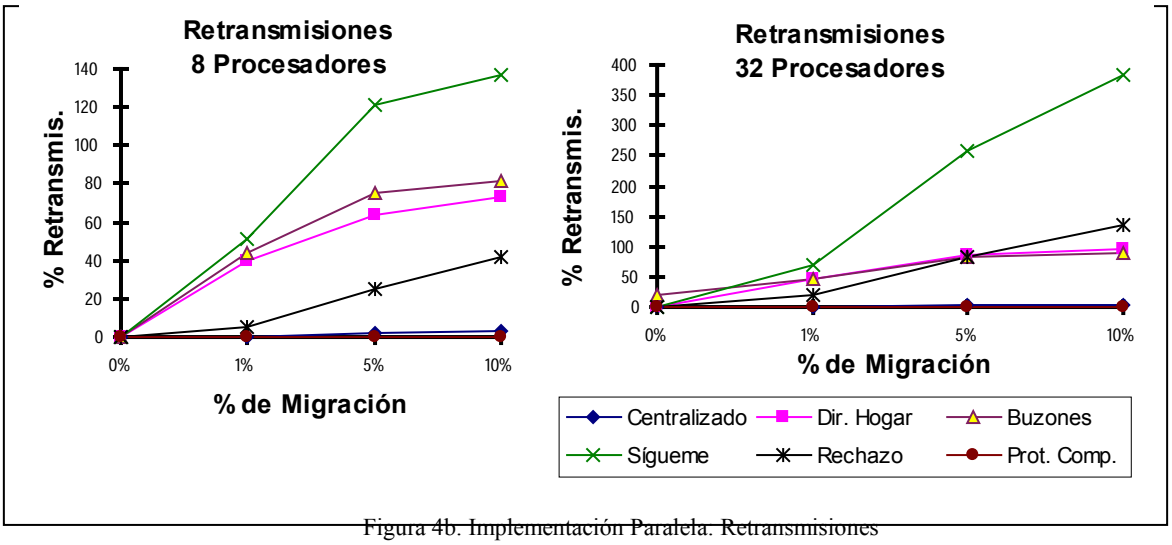

Figura 4b. Implementación Paralela. Retransmisiones

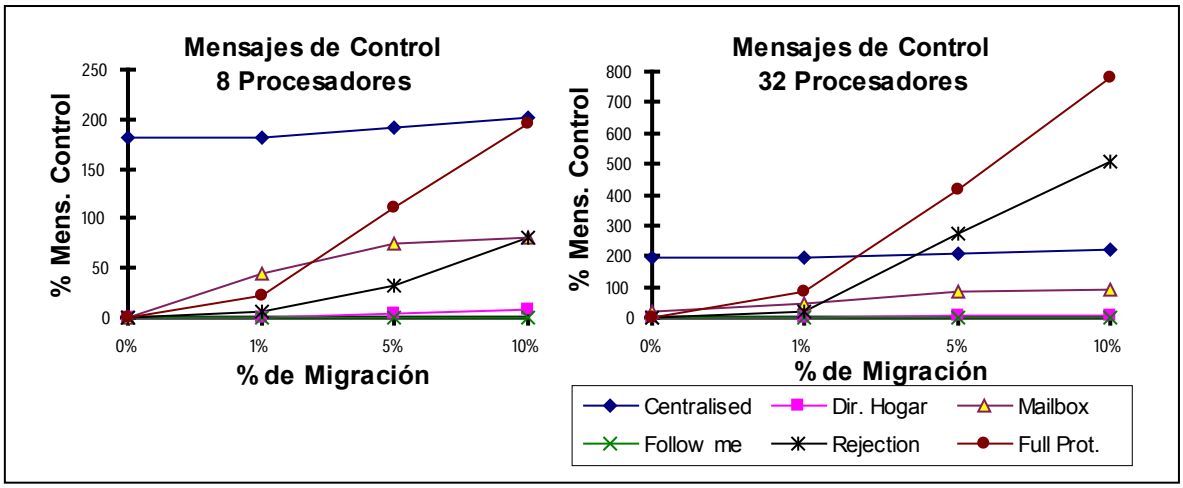

Figura 4c. Implementación Paralela: Mensajes de Control

5. Análisis de los Resultados

Respecto a la latencia se han obtenido cuatro tipos de comportamiento diferente en la simulación secuencial: (1) El Protocolo Completo tiene una latencia igual al tiempo de comunicación de enviar un mensaje desde un proceso hasta otro, sin migración, independientemente del número de procesadores y del porcentaje de migración; (2) el Esquema Centralizado tiene una latencia constante, independientemente del porcentaje de migración. Esta latencia es elevada en el caso de 8 procesadores, pero la diferencia respecto a la latencia de las demás políticas disminuye a medida que se añaden procesadores; (3) la latencia del algoritmo Sígueme empeora a medida que se añaden procesadores y a medida que incrementa el porcentaje de migración; y (4) el resto de los algoritmos (Enviar Primero a la Dirección Hogar, Buzones para Migración y Rechazo de Mensajes) tienen un comportamiento similar, consistente en presentar una latencia baja cuando el porcentaje de migración es bajo, y un punto a partir del cual la latencia se incrementa.

En la implementación paralela la latencia de los mensajes está fuertemente afectada por la carga de la red de comunicaciones. Los mensajes de usuario, las retransmisiones de los mensajes, el tráfico de control generado por cada política y la migración de procesos de usuario constituyen la carga de la red. Esto explica la diferencia en escala de los resultados para la implementación paralela (fig. 4a) y los resultados de la simulación secuencial (fig. 3a). Como la latencia de los mensajes depende de las retransmisiones, y estas retransmisiones se miden en la red de comunicaciones real, la política Sígueme resulta ser la peor en la mayoría de los casos (cantidad de procesadores y frecuencias de

migración). Más aún, la latencia de la política Protocolo Completo es afectada por la carga de la red de comunicaciones y los mensajes suspendidos debido a la migración en progreso de los procesos de usuario destinos.

Tanto la simulación secuencial como la implementación paralela muestran el incremento en la latencia cuando se utilizan más procesadores (figuras 3a y 4a). Aún así, la escalabilidad de las políticas (la forma en que una mayor cantidad de procesadores baja el rendimiento) que se ve en la implementación paralela es peor de la que se encuentra según la simulación secuencial. Tomando como referencia las medidas con 8 procesadores, la latencia de los mensajes cuando se utilizan 4 veces más procesadores en la implementación paralela (figura 4a) se incrementa por un factor entre 10 y 15 dependiendo de la política. Por contrapartida, la simulación secuencial muestra que utilizar 8 veces más procesadores decrementa el rendimiento en un factor entre 1.5 y 2.

Teniendo en cuenta la carga en la red de comunicaciones que produce cada política, hay un intercambio de comportamiento entre las retransmisiones de mensajes y los mensajes de control. Para cantidades de procesadores relativamente bajas, la cantidad de mensajes de control generadas por el Esquema Centralizado es inicialmente más alta que en cualquier otra política. De todas manera, aún en el caso de pocos procesadores y pocos procesos de usuarios la cantidad de mensajes de control que se generan en la política Protocolo Completo tiende a crecer exponencialmente en relación con la tasa de migración (figuras 3c y 4c). Por otro lado, la política Protocolo Completo no genera ninguna retransmisión (figuras 3b y 4b). Como la cantidad de mensajes de control que se generan en la política Protocolo Completo depende fuertemente de la interconexión entre los procesos de usuario , la figura 3d muestra esta relación para diferentes grafos de interconexión: total, *pipe* e hipercubo.

La escalabilidad en cuanto a la cantidad de retransmisiones de mensajes que se encuentra en todos los algoritmos es buena (figuras 3b y 4b). Aún así, la política Protocolo Completo tiene siempre los mejores resultados. Las retransmisiones de mensajes se incrementan en la implementación paralela para 32 procesadores comparada con los valores obtenidos para 8 procesadores (figura 4b), pero este incremento no es relevante y muestra, como en el caso de la simulación secuencial, una buena escalabilidad para este parámetro de rendimiento. Además, se encuentra que la política Rechazo de Mensajes en la implementación paralela con 8 procesadores se comporta de manera diferente para 32 procesadores (figura 4b).

La política Protocolo Completo tiene el mayor tiempo de reacción. Dependiendo de la implementación, en el Esquema centralizado y Enviar Primero a la Dirección Hogar, la nueva ubicación del proceso que migra tiene que ser actualizada (en el Servidor Central o en el procesador hogar respectivamente) antes de que se produzca la migración. También se puede enviar la nueva ubicación después de la migración. En el caso de la política de Buzones para Migración, el proceso de usuario debe esperar la llegada de los mensajes que ha requerido de su buzón. Usualmente hay a lo sumo una recepción pendiente y por lo tanto se debe esperar la llegada de un mensaje antes de estar habilitado para migrar. Los procesos de usuario en las políticas Sígueme y Rechazo de Mensajes reaccionan inmediatamente, todos los eventos se realizan localmente. Se deben cumplir una sucesión de pasos desde que se elige un proceso de usuario para migración hasta que se envía a otro procesador bajo la política Protocolo Completo. Las medidas que se han tomado en la implementación paralela muestran que el tiempo de reacción para la política Protocolo Completo varía entre el tiempo de 20 y 30 mensajes de usuario (el proceso de usuario podría enviar entre 20 y 30 mensajes mientras se está llevando a cabo el protocolo de migración).

Si un algoritmo no modifica adecuadamente el patrón de comunicación en la red, no será capaz de obtener el balance de las comunicaciones. La política Protocolo Completo, después del paso de intercambio de información, donde se genera una cantidad de mensajes que dependen de la conexión entre procesos de usuario, modifica el patrón de comunicaciones. La política Rechazo de Mensajes también lo cambia. Por otro lado, el patrón de comunicaciones permanece más o menos sin alteraciones en la política Sígueme. Independientemente de la ubicación de un proceso de usuario que ha migrado, las políticas Enviar Primero a la Dirección Hogar y Buzones para Migración mantienen la carga alrededor del procesador hogar y del procesador que contiene al buzón para migración respectivamente. El Esquema Centralizado produce una sobrecarga alrededor del Servidor Central.

6. Conclusiones y Líneas Abiertas

Para poder balancear la carga y las comunicaciones en un CPMD es necesario contar con un mecanismo de migración de procesos en tiempo de ejecución. Dicho mecanismo debe soportar que los procesos que migren reciban los mensajes destinados a ellos, independientemente de su ubicación en la red. Para manejar este problema, al cual hemos llamado el Problema de la Integridad de Mensajes, seis algoritmos con diferentes características se han desarrollado, estudiado por simulación secuencial e implementado sobre una máquina paralela.

La simulación secuencial nos ha dado una idea preliminar de las ventajas y desventajas del comportamiento intrínseco de cada algoritmo bajo distintas situaciones, respecto a los parámetros de prestaciones considerados, sin la influencia de factores externos.

La implementación paralela ha permitido considerar mas parámetros, como la carga de la red, y de obtener resultados mas representativos, al considerar condiciones reales de trafico de mensajes y de procesos.

Ninguno de los algoritmos básicos ha resultado ser suficientemente bueno para todas las situaciones, así que se han sugerido algunas mejoras. Sin embargo, se han descartado inicialmente algunos algoritmos como el Protocolo Completo, el Sígueme o el Esquema Centralizado. Como siguiente paso se deben desarrollar y estudiar nuevos algoritmos que combinen las ventajas de los algoritmos propuestos. Actualmente se están desarrollando índices que permitan caracterizar mejor los algoritmos desarrollados. Por último, se están desarrollando modelos formales para los algoritmos, lo cual permitirá predecir resultados sin necesidad de simulación.

7. Referencias

[1] Luque, E; Ripoll, A; Cortès, A; Margalef, T. **A Distributed Diffusion Method for Dynamic Load Balancing on Parallel Computers.** Proceedings of the EUROMICRO Workshop on Parallel and Distributed Processing, IEEE Computer Society, Enero, 1995.

[2] Smith, J. **A survey of Process Migration Mechanisms.** Operating Systems Review, ACM SIGOPS, Julio 1988, pp. 28-40.

[3] Zhu, W.; Goscinski, A.; Gerrity, G. **Process Migration in RHODOS.** Australian Defense Force Academy, Canberra, 1990.

[4] Lenoski, D; Weber, W. **Scalable Shared-Memory Multiprocessing.** Morgan Kaufmann Publishers, 1995. Capítulos 2, 3 y 5.

[5] Luque, E.; Senar, M.; Hernández, P.; Franco, D.; Heymann, E.; Moure, J.C. **Programming Environment for a Transputer Based Computer**. Future Generation Computer Systems 10, Elsevier 1994, p. 295-299.

[6] Luque, E.; Franco, D.; Heymann, E.; Moure, J.C. **TransComm: A Communication Microkernel for Transputers.** Proc. of the Fourth Euromicro Workshop on Parallel and Distributed Processing. IEEE Computer Society Press, Enero, 1996, pp. 147-153.

[7] Heymann, E. **Soporte para la Migración de Procesos en Computadores Paralelos de Memoria Distribuida.** MSc Tesis, Departamento de Informática. Universitat Autónoma de Barcelona, 1995.