Un Algoritmo para Balancear Dinámicamente las Tareas de un Programa en Sistemas Paralelos[1]

A. Cortés, A. Ripoll, M.A. Senar, P. Pons y E. Luque
Departamento de Informática
Unidad de Arquitectura de Ordenadores y Sistemas Operativos
Universidad Autónoma de Barcelona
08193 - Bellaterra (Barcelona), SPAIN
Phone No.: + 34 - 93 - 581 1990
e-mail: {a.cortes, a.ripoll, m.a.senar, p.pons, e.luque}@cc.uab.es

Resumen
DASUD ("Diffusion Algorithm Searching Unbalanced Domains") es un algoritmo totalmente distribuido que se ha desarrollado para realizar el balanceo dinámico de carga y resolver los problemas de desequilibrio de carga que se producen cuando se suponen tareas indivisibles. Dichos problemas los encontramos en los algoritmos de tipo "nearest-neighbour" (vecinos inmediatos) ya que, dada la naturaleza local de la información que manejan, pueden obtener situaciones localmente balanceadas que, sin embargo, muestren un elevado desbalanceo global. DASUD se ha comparado con dos de las estrategias más conocidas en la literatura dentro de la familia de los algoritmos de vecinos inmediatos: GDE ("Generalised Dimension Exchange") y SID ("Sender Initiated Diffusion"). Para realizar la comparación, se ha utilizado un extenso conjunto de distribuciones de carga iniciales que cubren, por una parte, un amplio rango en el desequilibrio inicial de las cargas, y por otra, diferentes formas de distribución de dichas cargas en los procesadores. Las topologías que se han utilizado son las de tipo anillo, hipercubos y toros, y el número de procesadores varió desde 8 hasta 128. Del análisis de los resultados obtenidos, se concluye que DASUD supera a las otras estrategias ya que ofrece el mejor compromiso entre el grado de balanceo final y el tiempo necesario para alcanzarlo.

Palabras clave: *Balanceo dinámico de carga, algoritmos de balanceo entre vecinos inmediatos, métodos de difusión, aplicaciones no uniformes.*

1. INTRODUCCIÓN

Para obtener el máximo rendimiento de una aplicación paralela es necesario que su carga computacional esté distribuida equitativamente entre todos los procesadores del sistema para minimizar así el tiempo de ejecución. Existen muchas aplicaciones de tipo científico (simulaciones de partículas/plasma, resolución de ecuaciones diferenciales parciales, integración numérica,…) donde la carga de trabajo asociada a una determinada tarea puede cambiar a lo largo de la ejecución de la misma y, en consecuencia, no puede ser estimada previamente. Para este tipo de problemas no uniformes, con unos requerimientos de cómputo y comunicación impredecibles a priori, es necesario el uso de algoritmos de balanceo dinámico de carga con objeto de distribuir eficientemente las tareas en tiempo de ejecución sobre los procesadores del sistema.

En los últimos años han ido apareciendo una serie de estrategias de balanceo dinámico de carga que eran escalables, portables y de fácil utilización [Cyb89] [Kum94] [Son94] [Wil93] [Xu97]. Sin embargo, el problema que presentan estas técnicas es que no garantizan que se alcance una situación de perfecto equilibrio. La figura 1.1(a) muestra la situación final en la que todos los métodos anteriores dejarían las cargas de 5 procesadores, $p_i = 0 \ldots 4$, conectados linealmente.

[1] Este trabajo ha sido realizado dentro del proyecto TIC 95-0868 financiado por la CICYT.

A pesar de que ninguno de los procesadores difiere en más de una unidad con cualquiera de sus vecinos, la situación global de balanceo es pobre: la diferencia de carga entre los procesadores extremos es de 4 unidades. La solución óptima en este caso sería la que refleja la figura 1.1.(b).

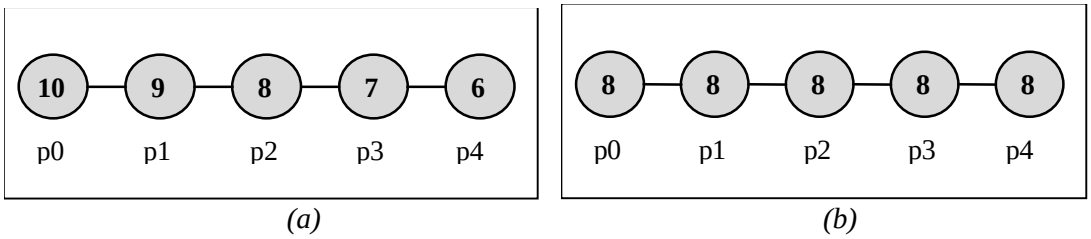


Figure 1.1 (a) Situación de desbalanceo global. (b) Situación de balanceo óptimo.

El objetivo de este trabajo es presentar un nuevo método que soluciona los problemas de desequilibrio de carga en los que incurren los métodos previos cuando se suponen tareas indivisibles, manteniendo las características de escalabilidad, que sea completamente local y apropiado para su uso en un entorno de propósito general de balanceo dinámico de carga. El algoritmo resultante, denominado DASUD ("Diffusion Algorithm Searching Unbalanced Domains") es distribuido, asíncrono (un procesador balancea su carga con independencia de lo que hagan los demás procesadores) y dinámico. Este algoritmo es capaz de detectar la mayoría de las situaciones desequilibradas ante las que fallan los métodos previos (como la mostrada en la figura 1.1), y realiza un conjunto de movimientos de carga que permiten alcanzar situaciones de equilibrio de carga perfecto (donde la máxima diferencia de carga entre cualquier pareja de procesadores es de una unidad). Las simulaciones que se han llevado a cabo demuestran los beneficios proporcionados por DASUD, ya sea en la calidad del balanceo de carga (la máxima diferencia de carga entre los procesadores), como en la eficiencia del algoritmo medida como el número de pasos y los movimientos de carga necesarios para transformar una distribución de carga inicial en una distribución equilibrada.

2. PLANTEAMIENTO DEL PROBLEMA

El problema del balanceo dinámico de la carga puede dividirse en las siguientes cuatro fases [Wil93]:

1. *Evaluación de la carga*. En esta fase debe estimarse el valor de carga de un procesador, valor que se utilizará para determinar la existencia o no de un desequilibrio de carga.
2. *Toma de decisión.* Fase que detecta la existencia de desbalanceo y evalúa el coste en el que se incurriría eventualmente al solucionarlo. El proceso de balanceo se iniciará si su coste no supera el coste asociado a la existencia del desbalanceo.
3. *Migración de carga.* Una vez decidido iniciar el balanceo, aquí se determinan los procesadores origen y destino que van a intervenir en el movimiento de carga. A los procesadores origen se les indica el número y el destino de las unidades de carga que deben mover.
4. *Selección de tareas*. El procesador origen selecciona las tareas más adecuadas equivalentes a las unidades de carga que se deben mover.

El objetivo de una estrategia de balanceo dinámico de carga consiste en balancear la carga para minimizar el tiempo de finalización de la aplicación. Esto supone que en el presente trabajo consideramos que una aplicación está compuesta por un conjunto de tareas distribuidas en el sistema, que son independientes y que pueden ser ejecutadas en cualquier procesador en cualquier orden. Además, dada la naturaleza no predecible de los requerimientos de las tareas, supondremos que todas las tarea tienen los mismos requerimientos de cómputo. Por lo tanto, la evaluación de la carga de un procesador se reduce a contar el número de tareas que tiene

pendientes de ejecución. De forma análoga, la fase de selección también queda simplificada pues no se establece ninguna distinción entre las diferentes tareas y se ignora cualquier aspecto relativo a la localidad de las mismas.

En general, los algoritmos de balanceo dinámico de carga toman sus decisiones en base a una cierta información local y gestionan los movimientos de carga con sus vecinos inmediatos (de ahí, la clasificación como algoritmos de vecinos próximos). Estos algoritmos se ejecutan iterativamente con la esperanza de que, tras sucesivas invocaciones, van a conseguir que el sistema consiga llegar a una situación de equilibrio global. De la literatura existente sobre el tema, sobresalen dos grandes métodos para iniciar las operaciones de balanceo. En el primer método se inicia el balanceo cuando la diferencia entre la carga de un procesador y el promedio local (es decir, la carga promedio del procesador y todos sus vecinos inmediatos) supera una determinada cota [Wil93]. En el segundo método, un procesador inicia las operaciones de balanceo cuando la diferencia relativa entre su carga y la de uno de sus vecinos supera una determinada cota [Xu97].

Cuando se ha constatado que es productivo realizar balanceo de carga, se debe determinar qué porción del exceso de carga va a transferirse y a quién. En la siguiente sección analizaremos dos de las estrategias clásicas que se han propuesto para la resolución de estas preguntas y presentaremos la nueva estrategia DASUD.

3. ANALISIS DE LOS ALGORITMOS DE VECINOS INMEDIATOS

Todos los métodos de vecinos inmediatos se basan en la aplicación a lo largo del tiempo de unas operaciones que, eventualmente, conducen a la situación de balanceo óptimo. El esquema general de funcionamiento que siguen estos métodos cuando su diseño es totalmente distribuido es el que muestra la figura 3.1, donde en el recuadro interno se incluiría el código correspondiente a la segunda y a la tercera fase mencionadas en la sección anterior.

```
(1) begin
(2) while (no converge) do begin
(3)   for all processors i
(4)     parbegin
(5)       Fase 2: Toma de decisión
          Fase 3: Migración de carga
(6)     parend
(7) end
```

Figura 3.1 Esquema general de un algoritmo totalmente distribuido de balanceo de carga

En esta sección, vamos a analizar las fases 2 y 3 correspondientes a los métodos GDE[2] ("Generalised Dimension Exchange"), SID ("Sender Initiated Diffusion") y DASUD ("Diffusion Algorithm Searching Unbalanced Domains"). Antes, pero, vamos a introducir una terminología básica necesaria para la comprensión de los distintos métodos.

El proceso de balanceo de carga tiene lugar en los instantes de tiempo 0, 1,… y así sucesivamente. En un cierto instante de tiempo *t*, cada procesador *i* tiene una determinada carga $w_i(t) \geq 0$. Definimos el vector de carga en el instante *t* como $w(t) = \left(w_1(t) \quad \ldots \quad w_n(t)\right)$, donde *n* es el número de procesadores del sistema. La suposición de que el número de tareas es

[2] Para referirnos a las distintas estrategias que se mencionan a lo largo del trabajo hemos utilizado siempre sus siglas en inglés

fijo se indica suponiendo que $L = \sum_{i=1}^{n} w_i(t)$ es un valor invariable en el tiempo. Definimos también $\overline{w}(t) = L/n$ como la carga promedio del sistema. El objetivo de cualquier algoritmo de balanceo de carga es distribuir las cargas de forma que en un cierto instante de tiempo t_j cada valor $w_i(t_j)$ esté "próximo" a $\overline{w}(t)$.

A continuación, vamos a describir el comportamiento de cada uno de los métodos indicados anteriormente.

3.1 Descripción del Algoritmo GDE

El algoritmo GDE es una estrategia síncrona que se basa en el método de Intercambio por Dimensiones ("DE: Dimension Exchange")[Cyb89]. De forma sintética, su funcionamiento se basa en que un procesador inicia las operaciones de balanceo cuando existe una diferencia relativa de carga entre la suya y la de uno de sus vecinos (ver figura 3.2). El proceso de balanceo se lleva a cabo dimensión a dimensión durante sucesivos pasos, donde una dimensión corresponde a un subconjunto de todas las parejas de procesadores conectados directamente. De esta forma, si el sistema tiene *k* dimensiones, cada procesador deberá realizar *k* pasos de balanceo, uno con el procesador con el que está conectado en la dimensión correspondiente. En cada uno de estos pasos, el procesador *i* balancea su carga con uno de sus vecinos repartiendo equitativamente la diferencia de carga mediante un parámetro de proporcionalidad fijo denominado parámetro de intercambio (λ, en la línea 4.7). El procesador actualiza el nuevo valor de carga resultante y repite la operación con el siguiente vecino. Este proceso se va repitiendo hasta que se alcance una situación satisfactoria de equilibrio de carga.

Algoritmos DE y GDE

```
(1) while (no converge) do begin
(2)   for all processors i
(3)       parbegin
(4)
      (4.1)    for each "dimensión/color"  k
      (4.2)    if existe un arco en la dimensión/color  k entre i y uno de sus vecinos j then
      (4.3)    begin
      (4.4)       wi(t) =carga  del procesador  i
      (4.5)       el valor actual de wi(t) se envía al procesador  j
      (4,5)       recibir el valor actual de wj(t) del procesador  j.
      (4.6)       if wi(t) > wj(t) then
      (4.7)             enviar λ *( wi(t) - wj(t)) unidades de carga al procesador j
      (4.8)    end

 (5)    parend
(6) end
```

Figura 3.2. Algoritmos de balanceo de carga por Intercambio de Dimensiones (DE) y por Intercambio de Dimensiones Generalizado (GDE)

En su concepción original el método fue propuesto para ser aplicado en topologías de tipo hipercubo n-dimensionales. En este caso, un "barrido" correspondía a la revisión por parte de un procesador de todos sus vecinos de cada una de las dimensiones del hipercubo. Aplicando técnicas de coloreado de grafos Hosseine et al. analizaron la forma de aplicar el método a estructuras de interconexión arbitrarias [Hos90]. Esta extensión se refleja en la figura 3.2

considerando *colores* en lugar de *dimensiones*. Por último, Xu et al. [Xu97] propusieron una nueva extensión al método, que denominaron algoritmo de Intercambio por Dimensiones Generalizado, donde en lugar de usar un mismo parámetro de intercambio para todas las topologías, proponían el uso de valores diferentes de λ en función de la topología subyacente.

Como mencionamos anteriormente, el algoritmo GDE supone que las cargas son divisibles infinitamente y, por lo tanto, se representan por números reales. Esta suposición es válida cuando la aplicación explota un paralelismo de granularidad fina. Para manejar un paralelismo de granularidad media o gruesa, sin embargo, el algoritmo debe ser capaz de trabajar con cargas que se representan por valores enteros no negativos. El algoritmo GDE puede ser adaptado a este fin redondeando por defecto el valor obtenido en la línea 4.7 ($\lfloor \lambda * (w_i(t) - w_j(t)) \rfloor$). Esta ha sido la versión utilizada en nuestro caso porque es la que garantiza su convergencia a un estado final estable aunque esté eventualmente desequilibrado [Mur97][Cor97].

3.2. Descripción de los Algoritmos SID y DASUD

Las estrategias SID y DASUD pertenecen a la categoría de algoritmos asíncronos de vecinos inmediatos y trabajan siguiendo un esquema de difusión, donde las cargas se "difunden" desde procesadores con exceso de carga a procesadores con defecto de carga de forma análoga al proceso físico del mismo nombre.

Entre todas las estrategias de difusión, la elección de SID se debe a que es uno de los métodos más sofisticados que soluciona varios problemas que suelen sufrir otros métodos basados en difusión [Sub94][Mur97][Cor97]. Este método fue inicialmente propuesto en [Wil93], suponiendo que las cargas eran infinitamente divisibles. El método puede adaptarse para manejar números enteros utilizando, como en el caso de GDE la función de redondeo por defecto. La demostración de la convergencia a un estado final estable de la adaptación de SID se puede encontrar en [Cor98].

El comportamiento de SID puede resumirse en el esquema algorítmico de la figura 3.3. Cada procesador ejecuta el mismo grupo de operaciones a lo largo del proceso de balanceo. Primero, cada procesador envía la información relativa a su carga local a todos sus vecinos y recibe de ellos la información recíproca (línea 4.1). Entonces, cada procesador calcula la carga promedio de su dominio (que le incluye a él y a todos su vecinos) y el valor $d_{ii}(t)$ (línea 4.2), que representa el exceso de carga de un procesador respecto a la media de su dominio. Si el procesador *i* tiene exceso de carga, $d_{ii}(t)$ será un valor negativo ($d_{ii}(t) < 0$). En caso contrario, si el procesador tiene defecto de carga, $d_{ii}(t)$ será un valor no negativo ($d_{ii}(t) \geq 0$).

Un procesador con exceso de carga ($d_{ii}(t) < 0$) repartirá parte del exceso de su carga sólo entre los vecinos con defecto. Por lo tanto, se calcula un nuevo valor $d_{ij}^{+}(t)$ asociado a cada uno de los vecinos con defecto de carga (línea 4.4) y se van acumulando todos estos déficits sobre el valor t_d. Este valor se utiliza para calcular la proporción de exceso de carga del procesador *i* que debe enviarse a cada uno de los procesadores *j* con déficit de carga ($P_{ij}(t)$ en la línea 4.5). El total de carga que el procesador *i* va a enviar al procesador *j* se calcula finalmente como $s_{ij}(t)$ y se envía al procesador correspondiente (línea 4.6). Una vez finalizado este proceso en todos los procesadores, no se producen más movimientos de carga en todo el sistema ya que $s_{ij}(t) = 0$ para todos los procesadores. Como ya se ha comentado con anterioridad, la situación final que deja SID es normalmente no balanceada. DASUD resuelve este desequilibrio de la

carga detectando dominios no balanceados, para lo cual se añaden el conjunto de sentencias que aparecen en la figura 3.3. En cada paso del algoritmo DASUD, un procesador *i* cualquiera puede efectuar movimientos de carga por una de las dos siguientes razones,:

i) si la ejecución de las líneas 4.1 a 4.6 (parte SID) produce algún movimiento de carga del procesador *i* a alguno de sus vecinos ($s_{ij}(t) \neq 0$ para algún vecino *j*), entonces no se ejecutarán las líneas 4.7 a 4.11 (extensión de DASUD).

ii) en caso contrario, es decir, si la ejecución de las líneas 4.1 a 4.6 (parte SID) no produce movimiento de carga ($s_{ij}(t) = 0$ para todos los vecinos), se ejecutarán las líneas 4.7 a 4.11 que corrigen el posible desbalanceo existente en el dominio.

Esta ejecución mútuamente excluyente de las dos partes del algoritmo es necesaria para demostrar que DASUD converge a un estado final estable [Cor98], y además permite mantener acotada la complejidad del algoritmo.

Para detectar la existencia de dominios no balanceados se evalúan cuatro parámetros adicionales (línea 4.7):

a) valor de la carga máxima del dominio: $w_i^{max}(t)$,
b) valor de la carga mínima en el dominio: $w_i^{min}(t)$,
c) valor de la carga máxima entre los procesadores vecinos: $w_{vi}^{max}(t)$,
d) valor de la carga mínima entre los procesadores vecinos: $w_{vi}^{min}(t)$.

Si la diferencia entre la carga máxima de los vecinos y la carga del propio procesador es mayor que uno (línea 4.8), existe una situación de desequilibrio y se van a mover cargas de acuerdo con alguna de las siguientes acciones:

- *Acción 1:* Si el procesador *i* es el procesador con carga máxima dentro de su dominio y todos sus vecinos tienen la misma carga entonces el procesador *i* distribuirá $\alpha = (w_i^{max}(t) - w_i^{min}(t) - 1)$ unidades de carga entre sus vecinos (línea 4.9).

- *Acción 2*: Si el procesador *i* es el procesador con carga máxima en el dominio pero no todos los vecinos tienen la misma carga, entonces se envía una unidad de carga a uno de los vecinos con carga mínima (línea 4.10).

- *Acción 3*: Si el dominio del procesador *i* no está balanceado pero el procesador *i* no es el más cargado entonces le ordenará a uno de sus vecinos con carga máxima que le envíe una unidad de carga a uno de los vecinos con carga mínima (línea 4.11).

Por último, aquel procesador que en ninguno de los pasos anteriores haya enviado alguna unidad de carga a otro procesador esperará la llegada de los mensajes de *mandato* generados por otros vecinos durante la acción 3. Si llegan tales mensajes, se ordenan y se manda una unidad de carga al procesador indicado en el primer mensaje de la lista (línea 4.12). La justificación detallada de todas las acciones que se llevan a cabo sobre los mensajes de mandato está fuera del ámbito de este artículo porque se necesitan para garantizar la convergencia de DASUD.

Algoritmo DASUD

(1) while (no converge) do begin
(2) for ALL processors i
(3) parbegin
(4)

Algoritmo SID

(4.1) Intercambiar información de carga con los vecinos
(4.2) Calcular: $\overline{w}_i(t)$ y $d_{ii}(t) = \overline{w}_i(t) - w_i(t)$
(4.3) if (el procesador i tiene exceso de carga) then
(4.4) evaluar el déficit de carga de los vecinos del procesador i:
if ($d_{ij}(t)>0$) then $d_{ij}^{+}(t) = d_{ij}(t)$; $td = td + d_{ij}^{+}(t)$
else $d_{ij}^{+}(t) = 0$
(4.5) y la proporción de su exceso de carga que debe ser movida: $P_{ij}(t) = d_{ij}^{+} \big/ td$
(4.6) enviar $s_{ij}(t) =$ floor (- ($P_{ij}(t) * d_{ii}(t)$) unidades de carga al procesador j
(4.7) if ($s_{ij}(t) = 0$ para todos los vecinos) then calcular: $w_i^{max}(t)$; $w_{vi}^{max}(t)$; $w_i^{min}(t)$; $w_{vi}^{min}(t)$;
(4.8) if $((w_i^{max}(t) - w_i^{min}(t)) > 1)$ then
begin
(4.9) if $(((w_i(t) = w_i^{max}(t))$ and $(w_{vi}^{max}(t) = w_{vi}^{min}(t)))$ then
distribuir el exceso de carga de una en una entre los vecinos con déficit de carga.
(4.10) if $((w_i(t) = w_i^{max}(t))$ and $(w_{vi}^{max}(t) \neq w_{vi}^{min}(t)))$ then
enviar una unidad de carga a uno de los vecinos con menor carga
(4.11) if $(w_i(t) \neq w_i^{max}(t))$ then
enviar un mensaje de mandato a uno de los procesadores de carga máxima para que envíe una unidad de carga a uno de los procesadores con carga mínima
end
if (no se ha enviado ninguna unidad de carga en alguno de los pasos previos) then recibir y ordenar en función del tiempo todos los mensajes de mandato y enviar una unidad de carga al procesador indicado en el primero de los mensajes de la lista y borrar el resto de mensajes de mandato.

(5) parend
(6) end

Figura 3.3 Algoritmo DASUD

4. RESULTADOS DE LAS SIMULACIONES

En esta sección vamos a comparar los tres algoritmos que fueron comentados en el apartado anterior. La comparación se hará en términos de la estabilidad y la eficiencia de cada uno de los algoritmos. La estabilidad (o calidad de balanceo) mide la capacidad de una cierta estrategia para llevar una cierta distribución de carga inicial a una situación final con una distribución global uniforme. La eficiencia mide el tiempo que necesita la estrategia para balancear la carga y llegar a una situación estable. Esta medida la hemos evaluado teniendo en cuenta, por una parte, el tiempo necesario para efectuar los movimientos de carga y, por otra parte, el número de pasos consumidos por la estrategia.

En la comparación entre las tres estrategias se utilizaron diferentes topologías sobre las que se aplicaron diferentes distribuciones de carga sintéticas que fuesen lo más representativas posibles. Las topologías utilizadas fueron las de tipo anillo, los hipercubos y los toros bidimensionales. El tamaño de estas topologías fue de 8, 16, 32, 64 y 128 procesadores (en el caso de los toros se usaron configuraciones de 9, 16, 36, 64 y 121 procesadores con objeto de manejar estructuras cuadradas).

Las distribuciones de carga sintéticas consistían en un vector de carga inicial *w(0)*. La suma de los componentes de *w(0)* proporcionaba la carga total del sistema, denominada *L*, lo que permitía evaluar *a priori* la distribución global uniforme que debería obtenerse en cada procesador al final del proceso de balanceo: $\lceil L/n \rceil$ o $\lfloor L/n \rfloor$, donde *n* es el tamaño de la topología. En nuestros experimentos consideramos una carga inicial de $L = 3000$ y las distribuciones de carga iniciales fueron clasificadas en dos grandes grupos: distribuciones *probables* y distribuciones *patológicas*.

Las distribuciones *probables* cubren todas aquellas situaciones que supuestamente se presentan en los sistemas reales donde la mayoría de los procesadores empiezan con una cierta carga inicial diferente de cero. En este caso, cada elemento $w_i(0)$ se obtuvo aleatoriamente siguiendo cuatro patrones de distribución distintos:

- Distribuciones iniciales con una variación del 25% respecto a la carga global promedio: $\forall i \;\; w_i(0) \in [\, L/n\text{-}0.25*L/n \,,\, L/n+0.25*L/n]$
- Distribuciones iniciales con una variación del 50% respecto a la carga global promedio: $\forall i \;\; w_i(0) \in [\, L/n\text{-}0.50*L/n \,,\, L/n+0.50*L/n]$
- Distribuciones iniciales con una variación del 75% respecto a la carga global promedio: $\forall i \;\; w_i(0) \in [\, L/n\text{-}0.75*L/n \,,\, L/n+0.75*L/n]$
- Distribuciones iniciales con una variación del 100% respecto a la carga global promedio: $\forall i \;\; w_i(0) \in [\, L/n\text{-}L/n \,,\, L/n+L/n]$

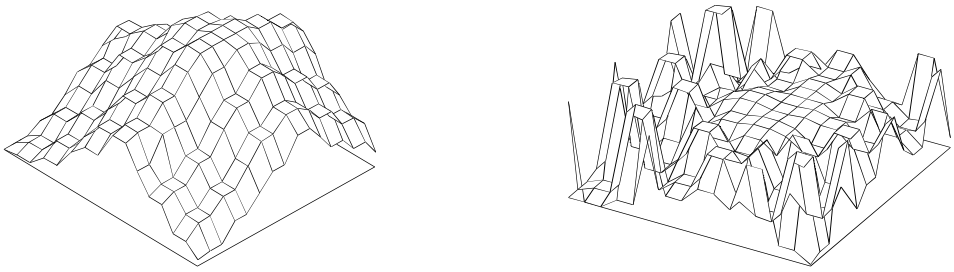
Dentro de cada uno de los cuatro grupos de patrones *probables* se usaron 10 distribuciones iniciales.

El grupo de las distribuciones patológicas fue usado con objeto de analizar el comportamiento de las estrategias bajo condiciones extremas en las distribuciones iniciales de carga. Estos casos son menos probables que aparezcan en situaciones reales pero fueron usados con objeto de hacer más completa la evaluación de las estrategies. Las distribuciones patológicas se subdividieron en cuatro grupos:

- Una distribución inicial en forma de pico, donde toda la carga se encuentra en un solo procesador: $w(0) = \begin{pmatrix} L & 0 & \dots & 0 \end{pmatrix}$.
- Una distribución inicial con el 25% de los procesadores sin carga.
- Una distribución inicial con el 50% de los procesadores sin carga.
- Una distribución inicial con el 75 % de los procesadores sin carga.

Además, tanto las distribuciones probables como las patológicas fueron colocadas sobre los procesadores siguiendo dos posibles formas: la de *montaña* y la de *cordillera*, definidas como:

1) Montaña, donde existe un punto central de máxima carga y a medida que nos alejamos del mismo la carga disminuye paulatinamente (ver figura 4.1(a)).
2) Cordillera, donde existen diferentes puntos localizados de carga elevada rodeados de zonas de menor carga (ver figura 4.1 (b)).



(a) (b)

Figura 4.1.Formas de colocación de la carga: Montaña (a), Cordillera (b)

Por lo tanto, en nuestros experimentos evaluamos no sólo la influencia de los valores iniciales de carga sino que también evaluamos la influencia que puede tener la forma en que esas cargas están colocadas sobre los procesadores.

En resumen, el número total de distribuciones de carga que probamos para una cierta configuración de procesadores fue de 87, obtenidas de la siguiente forma: 10 distribuciones *probables* * 4 patrones * 2 formas + 3 distribuciones *patológicas* * 2 formas + 1 distribución *patológica* en forma de pico.

En el proceso de simulación se consideró que los tres algoritmos estaban sincronizados globalmente con objeto de simplificar las tareas de programación y hacer los resultados más comprensibles. Además, las simulaciones se ejecutaron hasta que se llegaba a una situación de equilibrio global donde la estrategia correspondiente ya no efectuaba movimientos de carga entre dos pasos consecutivos. Por último, indicar que los valores que se usaron para el parámetro de intercambio (λ) del algoritmo GDE fueron de 0.5, 0.72 y 0.75 para hipercubos, anillos y toros, respectivamente. Estos valores fueron escogidos dado que son los que garantizan el mejor grado de convergencia en cada caso [Xu97].

4.1 Análisis de estabilidad

Como ya mencionamos anteriormente, la estabilidad es una medida de la habilidad de una cierta estrategia en llevar cualquier situación inicial hacia un estado de equilibrio perfecto. Dado que en nuestros experimentos suponemos valores de carga representados como números enteros, el estado final de equilibrio perfecto será aquel en el que la diferencia de carga entre dos procesadores cualesquiera del sistema sea cero o uno, dependiendo del valor de L y del número de procesadores.

Las figuras 4.2(a) y 4.2(b) muestran los resultados obtenidos en promedio en el análisis de estabilidad diferenciando las distribuciones *probables* y las *patológicas*. En el eje X aparecen las diferentes patrones aplicados y el eje Y se muestra la diferencia máxima de carga alcanzada por cada estrategia en el promedio de todos los casos analizados.

Como puede verse, SID fue siempre la estrategia que obtuvo peores resultados, siendo su diferencia máxima promedio mayor que la obtenida por GDE y DASUD en todos los casos. DASUD y GDE demostraron una cualidad adicional: para hipercubos y toros la diferencia máxima alcanzada se mantiene prácticamente constante con independencia del tamaño del sistema y del patrón inicial de la distribución; sólo en el caso de los anillos se observó un ligero incremento de la diferencia máxima cuando se usaban distribuciones iniciales más desbalanceadas (75% y 100%). En cualquier caso, DASUD fue superior también a GDE en todos los casos. Por ejemplo, GDE obtuvo en promedio diferencias máximas de 3.2 en toros y 3.3 en hipercubos cuando se usaron distribuciones probables, mientras que los valores de DASUD fueron de 1.8 y 1.4, respectivamente. En el caso de las distribuciones patológicas las

diferencias relativas entre las dos estrategias se mantuvieron aunque los valores absolutos de las diferencias máximas se incrementaron ligeramente.

Las tablas 4.1(a), 4.1(b) y 4.1(c) reflejan la desviación estándar respecto a la carga global promedio que obtuvieron cada una de las estrategias en el total de las simulaciones realizadas. De estos valores se observa que, en general, DASUD alcanza situaciones finales donde la desviación es muy pequeña (siempre menor que 1 en hipercubos y toros, y menor que 7 en anillos). Por contra, SID y GDE exhiben desviaciones estándar mayores en todos los casos: la de SID es 4 veces mayor que la de DASUD y la de GDE es entre 1 y 2 veces mayor.

De acuerdo con los datos reflejados en la figura 4.2 y en las tablas 4.1(a), 4.1(b) y 4.1(c), podemos concluir que, por un lado, DASUD obtiene en promedio valores más pequeños que SID y GDE en la diferencia máxima de carga y que, por otro lado, todos los procesadores del sistema alcanzan una situación similar y más próxima al estado de equilibrio perfecto.

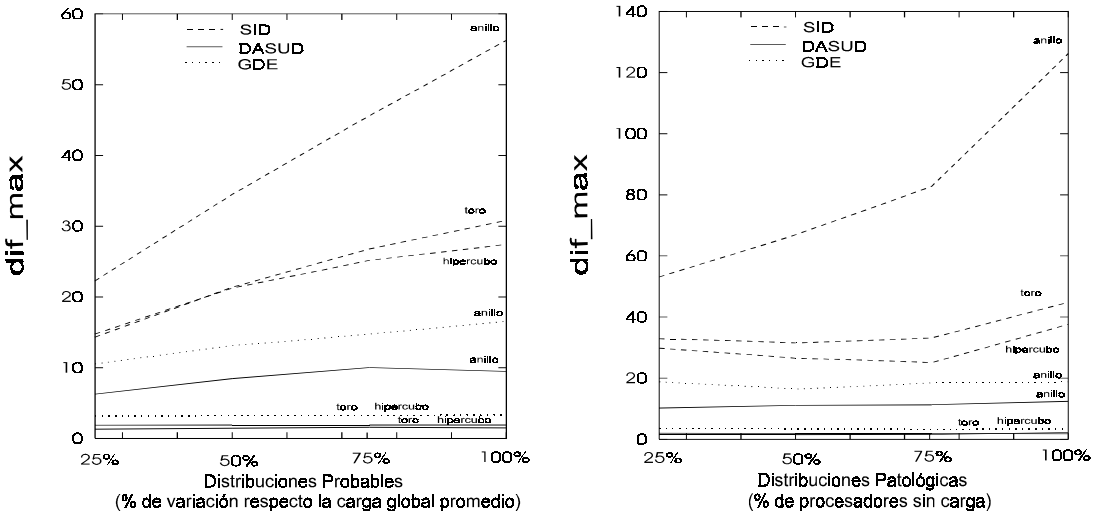


Figura 4.2.- Diferencia de carga máxima alcanzada por los algoritmos SID, GDE y DASUD considerando distribuciones probables (a) y distribuciones patológicas (b).

	Distribuciones probables					*Distribuciones patológicas*				
N. de proces.	8	16	32	64	128	8	16	32	64	128
Hipercubos	0.1	0.5	0.5	0.5	0.6	0.2	0.5	0.5	0.6	0.6
Toros	0.4	0.5	0.5	0.6	0.8	0.5	0.5	0.5	0.7	0.9
Anillos	0.5	1	2.3	4.4	5	0.6	1	2.3	4.8	6.2

Tabla 4.1(a). Desviación estándar obtenida en promedio por DASUD en las distribuciones probables y patológicas.

	Distribuciones probables					*Distribuciones patológicas*				
N. de proces.	8	16	32	64	128	8	16	32	64	128
Hipercubos	2.6	5.2	5.8	6.3	6.4	1.2	4.7	8.4	8.5	9.3
Toros	2.5	5.7.	5.8	7.4	8.2	2.1	4.4	11.3	13.6	14.4
Anillos	5.3	5.6	11	18.5	22.1	4.8	12.7	22.9	31.8	41.9

Tabla 4.1(b). Desviación estándar obtenida en promedio por SID en las distribuciones probables y patológicas.

	Distribuciones probables					*Distribuciones patológicas*				
N. de proces.	8	16	32	64	128	8	16	32	64	128
Hipercubos	0.6	0.7	0.8	0.9	0.9	0.7	0.8	0.8	1	1
Toros	0.5	0.6	0.8	1	1.2	0.5	0.6	0.8	1	1.5
Anillos	0.8	1.8	4.1	5.3	8.4	0.7	1.8	4.1	8.6	12.4

Tabla 4.1(c). Desviación estándar obtenida en promedio por GDE en las distribuciones probables y patológicas.

Por lo que respecta a la influencia de la forma usada en la distribución inicial de carga, observamos que la diferencia máxima y la desviación estándar siempre eran mayores en el caso de la colocación en forma de *Montaña*, aunque los valores obtenidos en este caso nunca fueron superiores al doble de los valores correspondientes a una colocación en forma de *Cordillera.* En este caso todas las estrategias demostraron un mismo comportamiento y ninguna de ellas exhibió unos resultados que tuviesen un interés especialmente reseñable.

4.2. Análisis de eficiencia

En este punto vamos a analizar la eficiencia de DASUD, SID y GDE. Esta medida refleja el tiempo que necesita una determinada estrategia en llegar a un estado de equilibrio. Para obtener ese "tiempo", en nuestras simulaciones medimos dos parámetros: el número de *pasos* de simulación consumidos por las estrategias hasta alcanzar un estado estable y el tiempo necesario para realizar los movimientos de carga a lo largo de todo el proceso de balanceo que denominamos *u*.

En un paso *s* del algoritmo de balanceo, representamos por *max_carga(s)* la cantidad máxima de carga que se mueve entre todos los procesadores. De acuerdo con nuestro paradigma de simulación síncrona, el paso *s* de simulación no va a finalizar hasta que *max_carga(s)* unidades de carga hayan sido movidas entre los procesadores correspondientes. Por lo tanto, la duración de cada paso depende directamente de *max_carga(s).* Para el cálculo de *max_carga(s)* se ha tenido en cuenta que el modelo de comunicación de la red de interconexión permite a un procesador comunicarse con todos sus vecinos simultáneamente. No obstante, esta propiedad no la aprovecha el algoritmo GDE porque en cada paso cada procesador sólo realiza intercambios de carga con uno de sus vecinos. El tiempo necesario para enviar una unidad de carga de un procesador a otro es el que denotamos como t_h. Por lo tanto, la suma de todos los valores de *max_carga(s)*, donde *s* varía de 1 hasta el número final de pasos de simulación,

multiplicada por t_h nos proporciona el tiempo total *u* requerido por el algoritmo para realizar el balanceo.

$$u = t_h * \sum_{s=1}^{s=ultimo\_paso} max\_carga(s)$$

Por simplicidad, en nuestro estudio suponemos que t_h es igual a uno.

La figura 4.3 muestra los valores correspondientes al parámetro *u* obtenidos por las distintas estrategias, tanto para distribuciones probables como patológicas. Para un número de procesadores concreto (eje X), la figura refleja el valor de tiempo que necesita cada estrategia en promedio para todos los patrones de distribución y todas las formas de colocación.

Como se puede observar, el tiempo que requirió DASUD fue moderadamente mayor que el tiempo que necesitó SID, siendo el tiempo de estas dos estrategias menor que el consumido por GDE. Los malos resultados que obtuvo GDE se pueden explicar porque, como mencionamos anteriormente, esta estrategia no aprovecha el posible solapamiento máximo de movimientos que la red de interconexión permite.

Para las distribuciones *probables*, el tiempo de DASUD fue aproximadamente el doble del de SID (GDE estaba algo por encima), aunque la diferencia máxima de carga que obtuvo SID en promedio fue 4 veces mayor que las de DASUD y GDE; para las distribuciones patológicas los tiempos de DASUD y GDE fueron, respectivamente, 1.3 y 3 veces mayores que los de SID, mientras que en este caso la diferencia máxima obtenida por SID fue, por ejemplo, 7 veces mayor que la obtenida por DASUD.

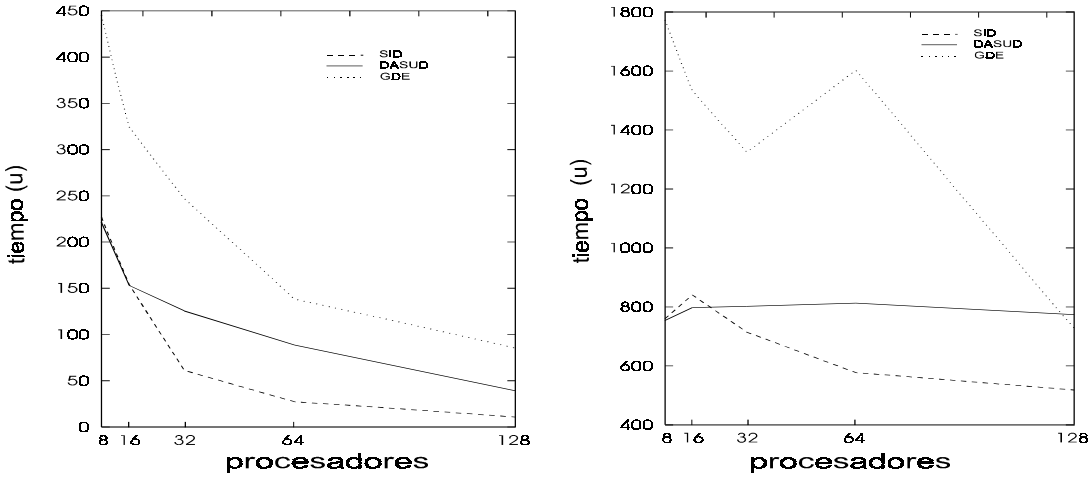


Figura 4.3. Resultados de eficiencia en términos de u's para: (a) distribuciones probables, y (b) distribuciones patológicas.

Lógicamente, al ser DASUD una extensión del algoritmo SID sus tiempos son mayores, aunque el tiempo adicional que consume es moderado en general. Si además tenemos en cuenta los valores correspondientes al análisis de estabilidad, podemos concluir que DASUD exhibe el mejor compromiso entre el grado de balanceo alcanzado y el tiempo necesario para alcanzarlo. En este sentido, DASUD obtiene un mejor balanceo final dada su capacidad para detectar dominios no balanceados, y consume un tiempo moderado porque mueve más carga en cada

paso y es capaz de aprovechar los movimientos solapados de carga que se producen en toda la red de interconexión.

De la figura 4.3 podemos deducir también un efecto positivo que muestran todas las estrategias para un tamaño de problema fijo (*L)* y para todas las distribuciones *probables*: el tiempo requerido para alcanzar el estado estable disminuye a medida que el tamaño de la topología aumenta. Esta característica se mantiene para cualquier porcentaje de variación en la carga inicial o cualquier forma de colocación. En principio esta característica era previsible dado que al aumentar el número de procesadores, la carga inicial que recibe cada uno de ellos es menor y, en consecuencia, se deberán mover menos unidades de carga.

El número de pasos consumido por cada estrategia se muestra en las tablas 4.2(a), 4.2(b) y 4.2(c). En el caso del algoritmo GDE la tabla 4.2(c) refleja el número de "barridos" efectuados, de acuerdo con la definición dada en la sección 3. Como se ve, GDE es normalmente la estrategia que necesita en promedio un menor número de pasos para todas las topologías sea cual sea su tamaño. SID necesita un número de pasos algo mayor y, por último, la estrategia que ha consumido más pasos es generalmente DASUD. El número de pasos que ha realizado DASUD fue casi siempre menor que el doble del número de pasos efectuados por GDE. Sin embargo, aunque GDE realiza menos pasos ("barridos") de simulación, el tiempo total para realizar el balanceo acaba siendo mayor por los motivos que comentamos anteriormente relativos al escaso solapamiento de movimiento de cargas.

Sólo en el caso de los anillos, y especialmente en las distribuciones patológicas, el número de pasos que ha necesitado DASUD es significativamente mayor que el de las otras estrategias. Esto se debe, en buena medida, a la mejor calidad de la solución alcanzada por DASUD. Mientras que SID y GDE terminan en un momento próximo al inicio y son incapaces de realizar más movimientos de carga, DASUD es capaz de realizar pasos adicionales en los que continua moviendo carga. Sin embargo, dado el reducido grado de conectividad que tienen las topologías de tipo anillo, el movimiento de cargas que efectúa DASUD es muy pequeño una vez alcanzado un cierto grado de balanceo, y la estrategia realiza un considerable número de pasos adicionales donde se mueve muy poca carga en cada uno de ellos.

	Distribuciones probables					*Distribuciones patológicas*				
N. de proc.	8	16	32	64	128	8	16	32	64	128
Hipercubo	9.66	13.4	14.6	16.7	14.3	9.85	14.2	19.42	22.4	22.5
Toro	7.5	13.35	17.06	19.95	19.15	9.28	14.1	27.4	36	44.57
Anillo	22.7	57.5	109.6	106.1	31.36	33.4	107	259	426.6	465.8

Tabla 4.2(a). Número de pasos efectuados en promedio por DASUD para distribuciones probables y patológicas.

	Distribuciones probables					*Distribuciones patológicas*				
N. de proc.	8	16	32	64	128	8	16	32	64	128
Hipercubos	8.71	9.93	9.11	7.4	4.4	8.57	11.28	12	9.85	6.2
Toros	8.3	10.13	15.26	12.45	3.47	9.2	11.57	23.85	29.71	29
Anillos	31.23	63	13.9	2.5	1	43.7	150.2	167.8	100.5	67.42

Tabla 4.2(b). Número de pasos efectuados en promedio por SID para distribuciones probables y patológicas.

	Distribuciones probables					*Distribuciones patológicas*				
N. de proces.	8	16	32	64	128	8	16	32	64	128
Hipercubos	6.25	6.87	7.6	7.9	6.4	7.5	8.75	9.6	10	9.25
Toros	10.2	10.15	9.67	9.9	10.27	12.25	12.75	13.25	14.25	16
Anillos	10.8	21.98	47.14	63.23	70	12.6	29.5	68.87	141.6	102.3

Tabla 4.2(c). Número de pasos efectuados en promedio por GDE para distribuciones probables y patológicas.

Por último, al analizar la influencia de la forma de la distribución de la carga inicial, observamos que todas las estrategias requieren aproximadamente el doble de pasos con la forma *Cordillera* que con la forma *Montaña*. Sin embargo, el tiempo en términos de *u*'s es ligeramente superior en el caso *Cordillera* que en el caso *Montaña*. Este fenómeno es atribuible al tipo de movimiento de carga que se produce en cada uno de los casos. Cuando la distribución inicial tiene forma de *Montaña* todos los movimientos de carga siguen una misma dirección, alejándose del procesador que constituye el pico de la montaña. Por otro lado, con la colocación de tipo *Cordillera* no todos los movimientos de carga que se llevan a cabo resultan productivos y todas las estrategias incurren en un cierto grado de trasiego de carga. En cualquier caso, al igual que en el análisis de la estabilidad, ninguna estrategia demostró en este aspecto particular un comportamiento especialmente remarcable.

5. CONCLUSIONES

En este trabajo hemos presentado un nuevo algoritmo DASUD ("Diffusion Algorithm Searching Unbalanced Domains"), pensado para realizar el balanceo dinámico de carga en sistemas paralelos. DASUD es una algoritmo asíncrono que supone una mejora significativa sobre los algoritmos existentes dentro de la familia de métodos de difusión, gracias a su capacidad para detectar dominios no balanceados. Es un algoritmo portable, de fácil implementación y de propósito general, y no está pensado, por lo tanto, para una topología o una aplicación en particular.

Hemos analizado la calidad del balanceo que obtiene DASUD, comparándolo con otros dos métodos significativos dentro de la familia de métodos de vecinos inmediatos: el método SID ("Sender Initiated Diffusion") y el método GDE ("Generalized Dimension Exchange"). El conjunto de experimentos realizados, usando un amplio y representativo espectro de distribuciones iniciales de carga, permite comprobar el grado de balanceo que alcanza cada una de las estrategias, así como el número de pasos que necesitan y el tiempo que consumen en el movimiento de carga.

Con estos experimentos, comprobamos que DASUD muestra un buen comportamiento en todos los casos probados, no siendo superado por ninguna de las otras estrategias por lo que se refiere al grado de balanceo final conseguido y, además, consume un tiempo relativamente parecido. DASUD supera a las estrategias SID y GDE al proporcionar el mejor compromiso entre el grado de balanceo global obtenido en el estado final y el número de iteraciones y el tiempo requeridos para alcanzar dicho estado.

6. REFERENCIAS

[Cor97] A. Cortés, A. Ripoll, M. A. Senar and E. Luque, Dynamic Load Balancing Strategy for Scalable Parallel Systems, PARCO'97.

[Cor98] A. Cortés, A. Ripoll, M.A.Senar, F. Cedó and E. Luque, On the convergence of SID and DASUD load-balancing algorithms, Technical Report, UAB, 1998.

[Cyb89] G. Cybenko, Dynamic Load Balancing for Distributed Memory Multiprocessors, J.Parallel Distributed Compt. 7, 1989, pp. 279-301.

[Hos90] S. H. Hosseini, B. Litow, M. Malkawi, J. McPherson, and K. Vairavan, Analysis of a Graph Coloring Based Distributed Load Balancing Algorithm, Journal of Parallel and Distributed Computing 10, 1990, pp. 160-166.

[Kum94] V. Kumar, A. Y. Grama and N. R. Vempaty, Scalable load balancing techniques for parallel computers. J. of Par. and Distrib. Comput., 22(1), 1994, pp. 60-79.

[Mur97] T. A. Murphy and J. G. Vaughan, On the Relative Performance of Diffusion and Dimension Exchange Load Balancing in Hypercubes, Procc. of the Fifth Euromicro Workshop on Parallel and Distributed Processing, PDP'97, January 1997, pp. 29-34.

[Son94] J. Song, A Partially Asynchronous and Iterative Algorithm for Distributed Load Balancing, Parallel Computing, vol. 20, 1994, pp. 853-868.

[Sub94] R. Subramain, I. D. Scherson, An Analysis of Diffusive Load-Balancing, In Proceedings of 6th ACM Symposium on Parallel Algorithms and Architectures, 1994.

[Wil93] M. Willebeek-LeMair, A. P. Reeves, Strategies for Dynamic Load Balancing on Highly Parallel Computers, IEEE Transactions on Parallel and Distributed Systems, vol. 4, No. 9, September 1993, pp. 979-993

[Xu97] C. Z. Xu and F. C. M. Lau, Load Balancing Parallel Computers - Theory and Practice, Kluwer Academic Publishers, 1997.