Balance dinámico de carga en Procesamiento Paralelo sobre Clusters No-Homogéneos

Armando De Giusti[1], Marcelo Naiouf[2], Laura De Giusti[3], Franco Chichizola[4]

Instituto de Investigación en Informática (III-LIDI) – Facultad de Informática – UNLP

RESUMEN

En este trabajo se discute el balance de carga estático y dinámico sobre arquitecturas de cluster no-homogéneo, analizando al mismo tiempo el Speedup paralelo teórico y el obtenido experimentalmente.
Se ha utilizado una combinación de 3 clusters interconectados, donde las máquinas dentro de cada cluster poseen procesadores homogéneos, pero diferentes entre clusters. De este modo el conjunto puede verse como un cluster heterogéneo de 25 procesadores o como un esquema multicluster con subconjuntos de procesadores homogéneos.
Se eligió una aplicación clásica (Parallel N-Queens) con un algoritmo de solución paralela en la que predomina el procesamiento sobre la comunicación, de modo de profundizar en los aspectos del balance de carga (estático o dinámico) sin una distorsión de los resultados producido por el overhead de comunicaciones.
Al mismo tiempo, se analizan tres formas de distribución de la carga en los procesadores (Estática Directa, Estática Predictiva y Dinámica por Demanda), estudiando en cada caso el Speedup paralelo y el desbalance de carga en función del tamaño del problema y los procesadores utilizados.

Palabras Clave: *Sistemas Paralelos. Arquitecturas de Cluster. Algoritmos paralelos. Balance de carga estático y dinámico. Speedup paralelo. Procesadores homogéneos y no-homogéneos.*

1 Investigador Principal CONICET. Profesor Titular D.E. Facultad de Informática UNLP. degiusti@lidi.info.unlp.edu.ar
2 Profesor Titular D.E. Facultad de Informática UNLP. mnaiouf@lidi.info.unlp.edu.ar
3 Becaria de Formación Superior UNLP. Jefe de Trabajos Prácticos Facultad de Informática UNLP. ldgiusti@lidi.info.unlp.edu.ar
4 Becario de Perfeccionamiento UNLP. Jefe de Trabajos Prácticos Facultad de Informática UNLP. francoch@lidi.info.unlp.edu.ar

Esta investigación es financiada por la CIC y la Fundación YPF.

1- Introducción

Arquitecturas de Cluster y MultiCluster

Un cluster es un tipo de arquitectura de procesamiento paralelo/distribuido que consiste de un conjunto de computadoras interconectadas que pueden actuar como una máquina única [20]. Las máquinas que componen un cluster pueden ser homogéneas o heterogéneas, lo que constituye un factor importante en el análisis de la performance que se puede obtener de un cluster como máquina paralela [1][4][8].

Una arquitectura multicluster consiste en la interconexión de dos o más clusters de modo de configurar una nueva máquina paralela. En ella, conceptualmente cada cluster interviniente puede verse como una máquina multiprocesador con determinados parámetros de performance que se integra en red con otras máquinas multiprocesador para obtener una arquitectura global única, capaz de realizar procesamiento paralelo combinando los recursos de cada cluster. La caracterización de los parámetros globales de performance de un multicluster resulta compleja, en función del número de clusters intervinientes, el grado de heterogeneidad de los procesadores en los mismos y las características del sistema de comunicaciones intercluster [14][19]. En ocasiones se trabaja con una combinación de clusters homogéneos, que se interconectan configurando un multicluster heterogéneo. Si bien el modelo del procesamiento puede simplificarse considerando un "supercluster" con un procesador por cluster interconectado, el modelo de comunicaciones sigue siendo complejo e incluso la comunicación inter-cluster puede tener un ancho de banda fijo o dependiente del tráfico general de comunicaciones (por ejemplo en clusters interconectados vía Internet) [26].

Esquema Master-Slave con arquitectura Multicluster

La utilización del paradigma Master-Slave con una arquitectura multicluster presenta al menos dos posibilidades:

- Si se utiliza un único procesador Master M, perteneciente a uno de los clusters del sistema, será muy importante caracterizar su performance y el tiempo de comunicación desde cualquier otro nodo del multicluster.
- Si se utiliza un procesador Master M_i por cluster, debe definirse un modelo de interacción entre estos M_i, de modo de controlar la actualización de información y las comunicaciones entre procesadores de diferentes clusters. El esquema de relación entre los M_i puede ser jerárquico o peer-to-peer. Nuevamente deben analizarse los diferentes tiempos de comunicación en juego.

La migración dinámica de datos y procesos entre nodos del multicluster tendrá un esquema diferente según cual de los dos modelos se adopte.

Balance de Carga en Arquitecturas heterogéneas

El balance de carga de una aplicación incide directamente en el speedup alcanzable y en el rendimiento del sistema paralelo [8][16]. En general, cuando se trabaja con clusters heterogéneos las diferentes potencias de cálculo de las máquinas intervinientes son un factor que puede computarse para analizar una distribución del trabajo a realizar.

En las clases de problemas de trabajo conocido (por ejemplo, multiplicación de matrices) puede obtenerse un balance de carga estático "predictivo" en función de la potencia de cálculo de los procesadores del multicluster; sin embargo, muchos problemas reales tienen una carga de trabajo variable o dinámica en función de los datos [2][9][18][27]. En estos casos es necesario ajustar dinámicamente la asignación de datos o procesos a los diferentes procesadores, a medida que la aplicación se ejecuta.

Nótese que cualquier fórmula de balance de carga "predictivo" debiera computar no sólo la potencia de cálculo, sino otros factores propios de la arquitectura tales como el tamaño y tiempo de acceso a los diferentes niveles de memoria de cada procesador [17][28]. En lo que sigue de este trabajo se ha considerado como factor principal sólo la potencia de cálculo de cada procesador.
Por otra parte, en un esquema multicluster en el que se resuelven aplicaciones con el paradigma Master-Slave, cualquier solución de balance dinámico implica un overhead de comunicaciones que será afectado por la complejidad del esquema de comunicación entre los nodos de los diferentes clusters, tal como se mencionó anteriormente.

Clases de problemas con carga de trabajo variable
Existen clases de problemas de paralelismo de datos en los que es posible realizar una asignación estática equilibrada de la carga de trabajo total. En estos casos, si se cuenta con una arquitectura heterogénea será posible definir una función predictiva $F(P_i,Wt)$ donde P_i es la potencia de cálculo del procesador i y Wt el trabajo total, con la cual distribuir los datos "a priori" entre los procesadores [21].
En el caso de tener una carga de trabajo variable en función de características de los datos (ej. ordenación de datos, reconocimiento de patrones en imágenes), no es posible tener una función predictiva que asegure el balance de carga entre los procesadores. Luego, será necesario tener alguna política de asignación dinámica que puede combinarse con una distribución inicial predictiva de un porcentaje de los datos totales [6][13][18].
Cualquier política de asignación dinámica implica algún grado de overhead de comunicaciones, que será más complejo de modelizar y predecir en una arquitectura multicluster heterogénea.

2- Caracterización de la clase de aplicaciones de interés.

Como se ha analizado en la introducción, existen diferentes ejes de investigación en los problemas de balance de carga dinámica sobre arquitecturas multicluster.
En este trabajo se ha fijado un modelo de arquitectura donde la heterogeneidad se presenta sólo entre máquinas de diferentes clusters y es posible aproximarla con una función de la potencia de cálculo de las máquinas de cada cluster.
Por otra parte, se ha elegido trabajar con el paradigma Master-Slave con un solo Master (ubicado normalmente en el cluster con mejores prestaciones de procesamiento).
Por último, en el trabajo experimental el foco se ha puesto en una clase de problemas en los que el tiempo de comunicación entre procesos Tc no es significativo frente al tiempo de procesamiento local Tp ($Tp >> Tc$). Esta restricción permite identificar más claramente las diferencias entre los esquemas de balance de carga estático y dinámico, sin superponer un overhead importante de comunicaciones.

3- Modelos de distribución de carga a estudiar y Speedup teórico alcanzable

Se utilizarán tres modos de implementar el paralelismo de datos:
- *Distribución estática directa (DEd)* donde la carga total de trabajo Wt se asignará en forma homogénea a los B procesadores de la arquitectura, de modo que cada procesador tendrá Wt/B, sin considerar la función $F(P_i,Wt)$.
- *Distribución estática predictiva (DEp)* donde la carga total de trabajo Wt se asignará al iniciar la aplicación a los B procesadores de la arquitectura, según la función de predicción $F(P_i,Wt)$.

- *Distribución dinámica por demanda (DDd)* donde un porcentaje *Li* de la carga total de trabajo *Wt* se asignará al iniciar la aplicación a los *B* procesadores de la arquitectura, según la función de predicción $F(P_i,Wi)$, y posteriormente cada procesador demandará más trabajo al Master, a medida que completa su tarea.

El valor de *Li* y la cantidad de trabajo adicional que se asigne a cada procesador cuando lo demande son parámetros de investigación experimental, que dependen de la aplicación y de la relación entre *Tp* y *Tc*.

El Speedup teórico alcanzable por la arquitectura multicluster será una función *G(Pi)*. Medir experimentalmente el Speedup real debe correlacionarse directamente con el grado de balance que se logre en la asignación del trabajo total *Wt* a lo largo de la ejecución de la aplicación.

4- Aporte de este trabajo

Se ha analizado un modelo Master-Slave con 3 clusters heterogéneos entre si, cada uno con 8 máquinas homogéneas, operando como un multicluster (*B*=24) con un procesador adicional como Master. La heterogeneidad de los procesadores fue contemplada a través de una función de la potencia de cálculo de los mismos.
Se estudió un caso problema que responde a la hipótesis *Tp* >> *Tc*, con las tres distribuciones de carga propuestas (*DEs*, *DEp*, *DDd*) para realizar paralelismo de datos, analizando especialmente el Speedup paralelo teórico y alcanzado en función de la potencia de cálculo de los procesadores, y el desbalance de carga en función de los parámetros *B*, *Wt*, P_i y *Li* que se mencionaron anteriormente.

5- Aplicación a la solución paralela sobre un multicluster heterogéneo del problema de las N-Queens

El problema de las *N*-reinas consiste en ubicar *N* reinas en un tablero de *N*x*N* de tal manera que ninguna de ellas ataque a otra [5][7][12]. Una reina ataca a otra si se encuentran en la misma diagonal, fila o columna.

5.1- Solución secuencial

Una solución inicial al problema de las *N*-reinas, mediante un algoritmo secuencial elemental, consiste en probar todas las combinaciones posibles de ubicación de las reinas en el tablero y quedarse con aquellas que son válidas, interrumpiendo la búsqueda en el momento en que esto no se cumple. Teniendo en cuenta que una combinación válida puede generar hasta 8 soluciones diferentes, las cuales son rotaciones de la misma, se puede reducir la cantidad de distribuciones que es necesario evaluar. En esto se basa el mejor algoritmo secuencial encontrado para este problema [3][23][24]. A continuación se realiza una breve descripción del mismo en un tablero de *N*x*N*.

El algoritmo realiza *N*/2 iteraciones, y en cada una de ellas ubica la reina en una posición diferente de la primera fila. Las *N*/2 posiciones restantes no son evaluadas debido a que son combinaciones simétricas de las anteriores.

A partir de la reina ubicada en la primera fila se determina el vector de posiciones válidas para la siguiente fila, y para cada una de ellas se determinan las soluciones que las mismas generan (Figura

1a). Para determinar la cantidad de soluciones a partir de la fila i (tal que toda fila j con j ≤ i tiene ubicada su reina), se determina el vector de posiciones válidas para la fila i+1, donde para cada una de ellas se vuelve a repetir este paso. Esto continúa hasta llegar a ubicar una reina en la última fila, o cuando no hay más posiciones válidas en una cierta fila (Figura 1b). Al llegar a ubicar una reina en la última fila se calcula la cantidad de soluciones diferentes que generan dicha combinación y su simétrica al ser rotadas 90º, 180º y 270º (Figura 1c).

```
main ()
{ cantSol:=0
  for (pos= 1..N/2)
    ubicarReina (1,pos,tablero)
    detPosVálida (posVálida,tablero,2)
    cantSol:=cantSol + detSol (2,posVálida,tablero)
}
```

(a)

```
function cantidadSoluciones (t)
{ if (rot(t,90)=t) or (rot(t,90)=sim(t)) then return (2)
  else if (rot(t,180)=t) or (rot(t,180)=sim(t)) then return (4)
       else return (8)
}

rot(t,g): retorna el tablero t rotado en g grados.
sim(t): retorna el tablero simétrico de t.
```

(c)

```
function detSol (fila, posVálida, tablero)
{ i:= posiciónVálida (posVálida)
  if (fila = N) and (i < N ) then
     ubicarReina (fila,i,tablero)
     return (cantidadSoluciones (tablero))
  else
    total:=0
    while (i < N)
       ubicarReina (fila,i,tablero)
       detPosVálida (nuevaPosVálida,tablero, fila+1).
       total:= total + detSol (fila+1,nuevaPosVáalida,tablero )
       i:= posiciónVálida (posVálida)
    return total
}

detPosVálida (p,t,f): determina el conjunto p de posiciones
válidas para la fila f en el tablero t.
posiciónVálida (p): retorna la primer posición válida dada en p.
```

(b)

Figura 1. Pseudocódigo de la solución secuencial

5.2- Solución paralela propuesta en función de los modelos de distribución de carga

Para la solución paralela de este problema, se ubica la reina de una o más filas y se obtienen todas las soluciones para esa disposición inicial. Cada procesador se encarga de resolver el problema para un subconjunto de éstas, de manera tal que el sistema completo trabaje con todas las posibles combinaciones de esas filas. Para esto hay que tener en cuenta dos temas:

- La forma de construir las combinaciones.
- Cómo distribuir las combinaciones entre las máquinas.

5.2.1- Cómo Construir las Combinaciones

Al trabajar con una arquitectura heterogénea la cantidad de trabajo (combinaciones) que debe resolver cada procesador varía según la relación que existe en cuanto a la potencia de cálculo. Para lograr distribuir el trabajo en forma equilibrada, es conveniente usar "grano fino", esto es, muchas combinaciones de poco trabajo cada una con el objetivo de poder nivelar el trabajo realizado por cada máquina resolviendo varias de estas [10]. Para esto se usan las tres primeras filas para formar cada una de las combinaciones a resolver.

De esta manera se obtienen N^3 combinaciones diferentes para distribuir entre todos los procesadores heterogéneos, siendo N el tamaño del tablero.

5.2.2- Cómo Distribuir las Combinaciones

5.2.2.1- Distribución Estática Directa (DEd)

En este tipo de distribución cada procesador m_i calcula al comenzar su ejecución cuáles son las combinaciones que debe resolver. Estas se distribuyen en forma cíclica entre las máquinas, de manera que cada m_i resuelve Wt/B combinaciones distribuidas en forma cíclica.

5.2.2.2- Distribución Estática Predictiva (DEp)

En este tipo de distribución cada procesador m_i determina al comenzar su ejecución cuáles son las combinaciones que debe resolver. Para esto realiza los siguientes pasos:

- Obtiene la cantidad relativa de combinaciones $cr_j \; \forall$ j = 1..*B*.

$$cr_j = \frac{P_j}{P_{MaquinaMenosPotente}}$$

- Determina la cantidad de combinaciones *A* que forman un bloque.

$$A = \sum_{j=1}^{B} cr_j$$

- Para cada bloque, calcula las combinaciones a realizar considerando que se asignan en forma cíclica a cada máquina m_j que no haya realizado cr_j combinaciones en ese bloque.

La Figura 2 muestra cómo se realiza la distribución suponiendo un sistema compuesto por seis máquinas heterogéneas (dos por cada cluster), en el cual las cantidades relativas de combinaciones son m_1=3, m_2=3, m_3=2, m_4=2, m_5=1 y m_6=1.

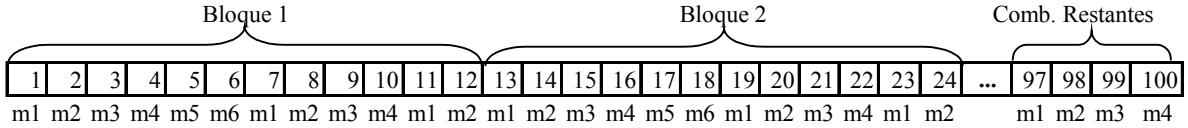

Figura 2: Distribución Estática Predictiva

La Figura 3 muestra el pseudocódigo del algoritmo completo para esta distribución:

```
main ()    //procesador 1
{ cantSol:=0
  while  (procesador 1 tenga combinaciones)
     determina la ubicación en la fila 1 (p1)
     determina la ubicación en la fila 2 (p2)
     determina la ubicación en la fila 3 (p3)
     cantSol:= cantSol + detSolParcial (p1,p2,p3,tablero)
  for (i=2..B)
     recv(cantOtroProc,i)
     cantSol:= cantSol + cantOtroProc;
 }
```

(a)

```
main ()   //procesador i con i > 1
{ cantSol:=0
  while  (procesador i tenga combinaciones)
     determina la ubicación en la fila 1 (p1)
     determina la ubicación en la fila 2 (p2)
     determina la ubicación en la fila 3 (p3)
     cantSol:= cantSol + detSolParcial (p1,p2,p3,tablero)
  send(cantSol,1)
 }
```

(b)

```
function detSolParcial (posFila1, posFila2, posFila3, tablero)
{ ubicarReina (1,posFila1,tablero)
  ubicarReina (2,posFila2,tablero)
  ubicarReina (3,posFila3,tablero)
  detPosVálida (posVálida, tablero,4).
  return detSol  (4, posVálida, tablero)
}

detPosVálida (p,t,f): determina el conjunto p de posiciones
válidas para la fila f del tablero t.
```

(c)

Figura 3. Pseudocódigo para DE_p. La función detSol es la descripta en la Figura 1 (b).

5.2.2.3- Distribución dinámica por demanda (DDd)

Dadas *C* combinaciones a calcular por *B* procesadores *slaves* y un *master*, el algoritmo realiza los siguientes pasos:

- El procesador master (m_0) reparte las combinaciones iniciales:
  - m_0 calcula la cantidad de combinaciones (*Ci*) a repartir inicialmente.

$$Ci = \frac{Wt \times Li}{100}$$

- m_0 obtiene la cantidad de combinaciones iniciales cc_i que le corresponden al procesador m_i de acuerdo a P_i

$$potTotal = \sum_{i=1}^{B} p_i \qquad cc_i = \frac{Ci \times P_i}{potTotal}$$

- m_0 distribuye las Ci combinaciones iniciales asignando cc_i combinaciones consecutivas a m_i, con $i = 1..B$.

▪ El procesador master (m_0) reparte las Cr combinaciones restantes, donde $Cr = C – Ci$.
- m_i solicita a m_0 más combinaciones para resolver cuando ha terminado su trabajo. Para cualquier $i = 1..B$.
- m_0 envía Cb combinaciones consecutivas al procesador m_i que hizo la solicitud, si aún hay trabajo por realizar.
- *Si* $Cb > 1$, m_0 decrementa en uno su valor .

La Figura 4 muestra cómo se realiza la distribución suponiendo un sistema compuesto por seis máquinas esclavas heterogéneas (dos por cada cluster), en el cual las potencias de cálculo relativas son m_1=3, m_2=3, m_3=2, m_4=2, m_5=1 y m_6=1. El valor de Li = 12 %, y el de Cb = 5.

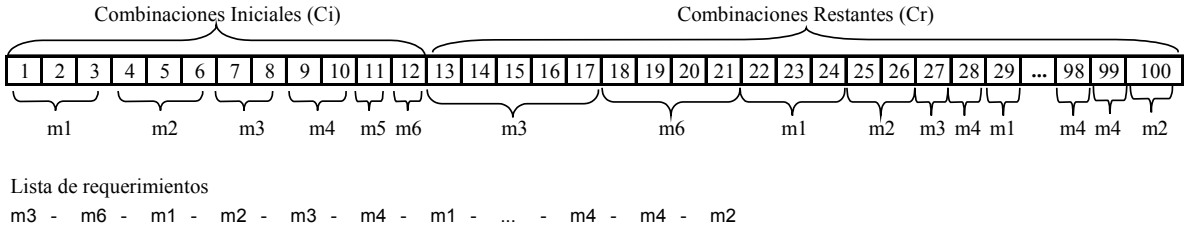

Figura 4: Distribución Dinámica por Demanda

La Figura 5 presenta el pseudocódigo del algoritmo para DD_d.

```
main ()   //procesador 0
{ cantSol:= 0
  for (i=1..B)
     determina  la primer combinación (pri)
     determina la última combinación (ult)
     send (pri,ult,i)
  while  (haya combinaciones)
     recv(pedido,i)
     determina  la primer combinación (pri)
     determina la última combinación (ult)
     send (pri,ult,i)
  for (i=1..B)
    send (fin,fin,i)
     recv (cantOtroProc,i)
     cantSol:= cantSol + cantOtroProc;
}
```

```
main ()    // procesador i con i > 0
{ cantSol:=0
  recv(pri,ult,0)
  while  (pri<>end)
     for (combinación= pri..ult)
        determina la ubicación en la fila 1 (combinación, p1)
        determina la ubicación en la fila 2 (combinación, p2)
        determina la ubicación en la fila 3 (combinación, p3)
        cantSol:=cantSol + detSolParcial (p1,p2,p3, tablero)
     send(pedido,i,0)
     recv(pri,ult, 0)
  send(cantSol,0)
}
```

Figura 5. Pseudocódigo para DD_d. La función detSolPartial es la descripta en la Figura 4.

6- Resultados Experimentales obtenidos

En esta sección se presentan las pruebas realizadas junto a los resultados obtenidos respecto de las métricas de speedup y desbalance que se describen a continuación.

6.1 Métricas Utilizadas

Para medir el desbalance de carga entre los procesadores que intervienen en una aplicación paralela, se calcula la diferencia de trabajo relativa obtenida a partir de la siguiente fórmula [4][5][24].

$$Desbalance = \frac{máximo_{i=1..B}(Trabajo_i) - mínimo_{i=1..B}(Trabajo_i)}{promedio_{i=1..B}(Trabajo_i)}$$

donde $Trabajo_i$ = *tiempo de la máquina*$_i$.

Para analizar el rendimiento del algoritmo en la arquitectura paralela se utiliza la métrica Speedup:

$$Speedup = \frac{TiempoSecuencial}{TiempoParalelo}$$

En el caso de una arquitectura heterogénea el "*Tiempo Secuencial*" está dado por el tiempo del mejor algoritmo secuencial ejecutado en la máquina con mayor potencia de cálculo [1][8][11].

Para evaluar cuan bueno es el speedup obtenido se compara con el speedup teórico de la arquitectura sobre la cual se está trabajando. El mismo considera la potencia de cálculo relativa de cada máquina con respecto a la potencia de la máquina más potente [25]. El speedup teórico se calcula por medio de la siguiente fórmula:

$$SpeedupTeórico = \sum_{i=1}^{B} P_i$$

donde

B es la cantidad de máquinas que componen la arquitectura utilizada.
P_i es la potencia de cálculo relativa de la máquina *i* con respecto a la potencia de la mejor máquina. Esta relación se expresa en la fórmula a continuación:

$$P_i = \frac{tiempoSecuencial(máquinaMasPotente)}{tiempoSecuencial(m_i)}$$

6.2 Experimentos

La experimentación se realizó sobre una arquitectura multi-cluster compuesta por tres clusters:

- un cluster homogéneo de 8 Celeron 2 Ghz, de 128 Mb de memoria.
- un cluster homogéneo compuesto por 8 Duron 800Mhz con 256 Mb de memoria.
- un cluster homogéneo compuesto por 8 Pentium III 700 Mhz, 256 Mb de memoria.

La comunicación dentro de cada cluster se hace por medio de una red Ethernet, y se utiliza un switch para la comunicación entre clusters.

El lenguaje utilizado para las implementaciones es C junto con la librería MPI para manejar las comunicaciones entre procesos [22].

Las pruebas se realizaron utilizando las 24 máquinas agregando una para la distribución dinámica que actúa como master, y con diferentes tamaños de tableros (N= 17, 18, 19, 20 y 21).

En el caso de la distribución dinámica, se probó con diferentes porcentajes de reparto inicial (*Li*) y con diferente cantidad inicial de combinaciones en los bloques a repartir (*Cb*).
Li = 0, 25, 50 y 75.
Cb = 1, 5, y 10.

6.3 Resultados

Los datos expuestos en la Tabla 1 presentan el porcentaje de desbalance de carga producido por el algoritmo para las distribuciones *Estática Directa, Estática Predictiva* y *Dinámica por Demanda* con diferentes valores de *Li* y *Cb*. En el Gráfico 1 pueden visualizarse algunos de estos resultados.

Tamaño	Estática Directa	Estática Predictiva	Dinámica por Demanda											
			0% - 1	0% - 5	0% -10	25% - 1	25% - 5	25% -10	50% - 1	50% - 5	50% -10	75% - 1	75% - 5	75% -10
17	93%	45%	11%	11%	10%	11%	9%	61%	8%	13%	113%	39%	39%	47%
18	98%	23%	9%	10%	11%	11%	11%	11%	11%	8%	67%	35%	35%	40%
19	130%	58%	8%	9%	9%	9%	9%	9%	9%	8%	68%	20%	20%	37%
20	88%	32%	6%	6%	6%	8%	7%	37%	8%	6%	49%	36%	36%	43%
21	110%	29%	7%	6%	6%	5%	6%	4%	7%	4%	36%	30%	30%	31%

Tabla 1: Porcentaje de desbalance para cada prueba.

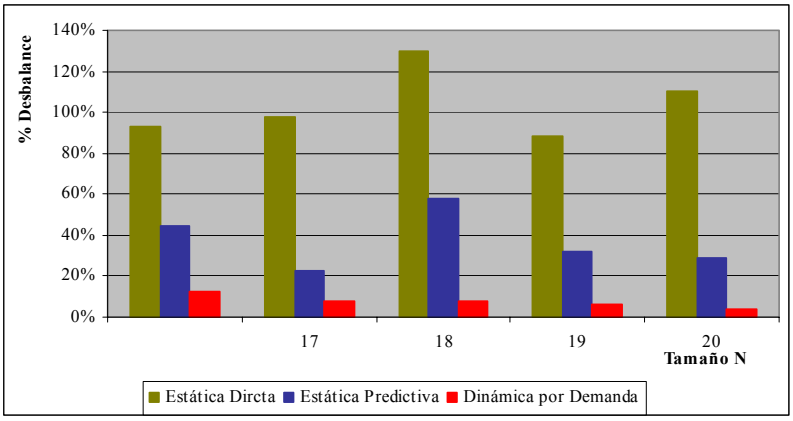

Gráfico 1: Desbalance de carga de las Distribuciones Estática Directa, Estática Predictiva y Dinámica por Demanda (con *Li*=25 y *Cb*=1). Para *N*=17, 18, 19, 20, 21.

La Tabla 2 presenta el speedup obtenido para cada una de las pruebas mencionadas anteriormente junto con el speedup óptimo (o teórico) calculado para está combinación de máquinas. La Tabla 3 muestra el tiempo total en cada prueba.

Tamaño	Óptimo	Estática Directa	Estática Predictiva	Dinámica por Demanda										
				0% - 1	0% - 5	0% -10	25% - 1	25% - 5	25% -10	50% - 1	50% - 5	50% -10	75% - 1	75% - 5	75% -10
17	16	9.42	13.49	14.78	14.82	14.80	14.83	15.16	10.22	15.16	14.61	7.72	12.19	12.19	12.19
18	16	9.28	14.25	15.12	14.98	14.89	14.89	14.89	14.91	14.90	15.12	9.82	12.43	12.43	12.34
19	16	8.09	12.80	15.13	15.12	15.15	15.22	15.14	15.15	15.17	15.20	9.85	13.89	13.89	12.27
20	16	9.82	13.61	15.39	15.27	15.32	15.26	15.30	12.02	15.25	15.43	11.09	12.39	12.38	11.89
21	16	8.96	13.56	15.36	15.40	15.36	15.42	15.45	15.68	15.39	15.54	12.03	12.73	12.73	12.67

Tabla 2: Speedup.

Tamaño	Estática Directa	Estática Predictiva	Dinámica por Demanda											
			0% - 1	0% - 5	0% -10	25% - 1	25% - 5	25% -10	50% - 1	50% - 5	50% -10	75% - 1	75% - 5	75% -10
17	6.36	4.44	4.05	4.04	4.05	4.04	3.95	5.86	3.95	4.10	7.76	4.91	4.91	4.91
18	46.88	30.51	28.75	29.02	29.20	29.21	29.21	29.16	29.18	28.76	44.30	35.00	34.99	35.23
19	413.18	261.28	220.96	221.11	220.63	219.63	220.80	220.67	220.45	219.94	339.58	240.72	240.72	272.56
20	2735.21	1973.48	1745.32	1759.14	1752.54	1759.64	1755.23	2234.18	1760.48	1740.82	2422.17	2168.29	2169.02	2259.31
21	25296.55	16710.35	14752.70	14720.81	14750.63	14699.02	14673.81	14450.14	14721.65	14585.20	18846.13	17809.43	17810.01	17886.58

Tabla 3: Tiempo Total del Algoritmo.

En el Gráfico 2 puede verse el speedup obtenido con cada uno de los algoritmos de distribución para algunas de las pruebas de la Tabla 2, junto al speedup óptimo de esta arquitectura.

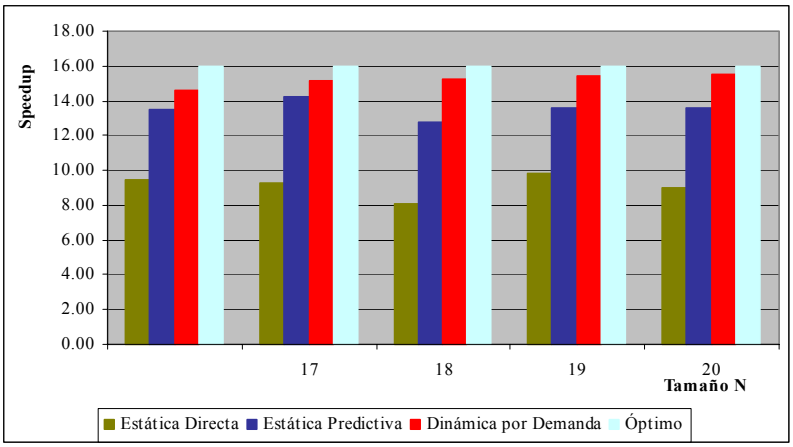

Gráfico 2: Speedup de las Distribuciones Estática Directa, Estática Predictiva y Dinámica por Demanda (con *Li*=50 y *Cb*=5) y Óptimo, para *N*=17, 18, 19, 20, 21.

En los resultados obtenidos se observa que el speedup logrado con la distribución dinámica por demanda se incrementa levemente a medida que aumenta el tamaño de *N*, siendo en todos los casos cercano al óptimo. Por otro lado, se puede ver que la diferencia es notoria con el speedup logrado por el algoritmo que distribuye en forma estática directa, y un poco menor con la que lo hace en forma estática predictiva.

7- Conclusiones y Líneas de Trabajo

Analizando los resultados obtenidos en el trabajo experimental podemos sintetizar algunas conclusiones:

- La solución paralela pura (sin tener en cuenta la distribución del trabajo) para el tipo de problemas donde *Tp*>>*Tc*, en particular el de *N*-Reinas requiere mínima comunicación entre máquinas, lo que hace esencial la elección de la distribución de datos entre los clusters, para alcanzar un Speedup cercano al óptimo.

- Naturalmente los algoritmos que tienen en cuenta la potencia de cálculo de cada máquina para la distribución del trabajo se comportan mejor que la distribución *Estática Directa*. Esta mejora se expresa claramente en el Balance de Carga y Speedup.

- Entre los algoritmos que tienen en cuenta la potencia de cálculo, se puede observar aquel que distribuye en forma dinámica logran distribuir el trabajo de manera más balanceada entre todas las máquinas (como se observa en el Gráfico 1), sin afectar demasiado el tiempo final de la ejecución (como lo indica el speedup en el Gráfico 2 y los datos de la Tabla 3).

- En la distribución dinámica el Speedup obtenido es muy cercano al óptimo de acuerdo a la arquitectura paralela que se utiliza en este caso.

Actualmente se está trabajando con un cuarto cluster, con máquinas sensiblemente más potentes que los tres clusters utilizados en el trabajo experimental expuesto.
Asimismo se realizan pruebas con clusters externos a la UNLP, en particular de la UN Sur (Bahía Blanca), de la UN Comahue (Neuquen), de la UA Barcelona(España) y la Universidad Católica del Salvador (Brasil).

8- Referencias

[1] Al-Jaroodi J, Mohamed N, Jiang H, Swanson D. "Modeling Parallel Applications Performance on Heterogeneous System". IEEE Computer Society, 2003.

[2] Baiardi F, Chiti S, Mori P, Ricci L. "Integrating Load Balancing and Locality in the Parallelization of Irregular Problems". Future Generation Computer Systems, Elsevier Science B.V., Vol 17, 2001, pp 969-975.

[3] Bernhardsson B. "Explicit Solution to the n-queens Problems for all n". ACM SIGART Bulletin,2:7,1991.

[4] Bohn C, Lamont G. "Load Balancing for Heterogeneous Clusters of PCs". Future Generation Computer Systems, Elsevier Science B.V., Vol 18, 2002, pp 389-400.

[5] Bruen A, Dixon R. "Then n-queens Problem. Discrete Mathematics". 12:393-395, 1997.

[6] Campos L, Scherson I. "Rate of Change Load Balancing in Distributed and Parallel System". Proceeding of the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, San Juan, Puerto Rico, Pages 701-707. 1999.

[7] De Giusti L, Novarini P, Naiouf M, De Giusti A. "Parallelization of the N-queens problem. Load unbalance analysis". Workshop de Procesamiento Paralelo y Distribuido (WPPD), Congreso Argentino de Ciencias de la Computación (CACIC'03), 2003.

[8] Donaldson V, Berman F, Paturi R. "Program Speedup in a Heterogeneous Computing Network". Journal of Parallel and Distributed Computing 21:3 (6/1994), 316-322.

[9] Dongarra J, Foster I, Fox G, Gropp W, Kennedy K, Torczon L, White A. "The Sourcebook of Parallel Computing". Morgan Kauffman Publishers. Elsevier Science, 2003.

[10] Goldman. "Scalable Algorithms for Complete Exchange on Multi-Cluster Networks". In: CCGRID'02, IEEE/ACM, Berlin, p. 286 - 287, 2002.

[11] Grama A, Gupta A, Karypis G, Kumar V. "Introduction to Parallel Computing". Second Edition. Pearson Addison Wesley, 2003.

[12] Hedetniemi S, Hedetniemi T, Reynolds R. "Combinatorial problems on chessboards: II". Chapter 6 in Domination in graphs: advanced topic, pag 133-162, 1998.

[13] Hui C, Chanson S. "Improve Strategies for Dynamic Load Balancing". IEEE Concurrency, pages 58-67. 1999.

[14] Jiang, Yeung. "Scalable Inter-Cluster Communication System for Clustered Multiprocessors". 1997.

[15] Jordan H, Alaghband G. "Fundamentals of Parallel Computing". Prentice Hall, 2002.

[16] Leopold C. "Parallel and Distributed Computing. A survey of Models, Paradigms, and Approaches". Wiley Series on Parallel and Distributed Computing. Albert Zomaya Series Editor, 2001.

[17] Menascé D, Almeida V. "Cost-Performance Analysis of Heterogeneity in Supercomputer Architectures". Proc. ACM-IEEE Supercomputing'90 Conference, New York, Nov 1990.

[18] Naiouf M. "Procesamiento Paralelo. Balance Dinámico de Carga en Algoritmos de Sorting". Tesis Doctoral. Universidad Nacional de La Plata, 2004.

[19] Ogura S, Nakada H, Matsuoka S. "Evaluation of the inter-cluster data transfer on Grid environment". Proceedings of CCGrid 2003 , pp. 374-381, May 2003.

[20] Pfister G. "In Search of Clusters". Prentice Hall, 2nd Edition, 1998.

[21] Ross K, Yao D. "Optimal Load Balancing and Scheduling in a Distributed Computer System". Journal of Association for Computing Machinery, 38 (3): 676-690.1991.

[22] Snir M, Otto S, Huss-Lederman S, Walker D, Dongarra J. "MPI: The Complete Reference". Cambridge, MA: MIT Press, 1996. Available in web site: http://www.netlib.org/utk/papers/mpi-book/mpi-book.html.

[23] Somers J. “The N Queens Problem a study in optimization”. www.jsomers.com/nqueen_demo/nqueens.html.
[24] Takaken, “N Queens Problem (number of Solutions)”. http://www.ic-net.or.jp/home /takaken /e/queen/.
[25] Tinetti F. “Cómputo Paralelo en Redes Locales de Computadoras”. Tesis Doctoral. Univ. Autónoma de Barcelona, 2004. https://lidi.info.unlp.edu.ar/~fernando/publis/publi.html
[26] Vaughan F, Grove D, Coddington P. “Communication Performance Issues for Two Cluster Computers”. Proceedings of the twenty-sixth Australasian computer science conference on Conference in research and practice in information technology, p.171-180, February 01, 2003, Adelaide, Australia.
[27] Watts J, Taylor S. “A Practical Approach to Dynamic Load Balancing”. IEEE Transactions on Parallel and Distributed Systems, 9(3), March 1998, pp. 235-248.
[28] Zhang X, Yan Y. “Modeling and Characterizing Parallel Computing Performance on Heterogeneous Networks of Workstations”. Proceeding of the 7th Symposium on Parallel and Distributed Processing. 1995.