Estudio del overhead en la migración de algoritmos paralelos de cluster y multicluster a GRID.

Ismael Rodriguez [1], Adrian Pousa [2], Jose Pettoruti [3], Franco Chichizola [4], Marcelo Naiouf [5], Laura De Giusti [6], Armando De Giusti [7]

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

Abstract

An experimental work is presented, in which a multicluster application has been migrated. In this application, communications among processes/processors was handled directly from MPI to the same architecture with Grid support (Globus Toolkit 4.04) in order to study the overhead generated by the incorporated middleware.
The N-queens solution with static distribution has been chosen and two clusters interconnected with Grid support, including the GridFTP, RFT, GRAM, WS-GRAM, RLS, and MDS services, were configured. Later, the response time for the N variable was studied, analyzing the speedup and efficiency.
The most interesting result is the overhead independence generated by the Grid middleware when scaling the problem.
Keywords: *Clusters. Grid. Grid Services. Parallel Algorithms. N-Queens.*

Resumen

Se presenta un trabajo experimental, en el que se migró una aplicación multicluster en la que las comunicaciones entre procesos/procesadores se manejaba directamente desde MPI a la misma arquitectura con soporte Grid (Globus Toolkit 4.0.4) con el objetivo de estudiar el overhead generado por el middleware incorporado.
Se eligió la solución de N reinas con distribución estática y se configuraron dos clusters interconectados con el soporte para Grid que incluía los servicios GridFTP, RFT, GRAM, WS-GRAM, RLS, MDS. Posteriormente se estudió el tiempo de respuesta para N variable, analizando el speedup y la eficiencia.
El resultado de mayor interés es la independencia del overhead generado por el middleware de Grid al escalar el problema.
Palabras Clave: *Clusters. Grid. Servicios Grid. Algoritmos paralelos. N-Reinas.*

VI Workshop de Procesamiento Distribuido y Paralelo.

[1] Becario Alumno Telefónica e III-LIDI Aux. Docente de la Facultad de Informática UNLP. ismael@lidi.info.unlp.edu.ar
[2] Becario Alumno Telefónica e III-LIDI Aux. Docente de la Facultad de Informática UNLP. apousa@ lidi.info.unlp.edu.ar.
[3] Becario Alumno III-LIDI. josep@ lidi.info.unlp.edu.ar.
[4] Becario de Doctorado del CONICET. Profesor Adjunto de la Facultad de Informática UNLP. francoch@lidi.info.unlp.edu.ar.
[5] Profesor Titular D.E. Facultad de Informática UNLP. mnaiouf@lidi.info.unlp.edu.ar.
[6] Profesor Adjunto de la Facultad de Informática UNLP. ldgiusti@lidi.info.unlp.edu.ar
[7] Investigador Principal CONICET. Profesor Titular de la Facultad de Informática UNLP. degiusti@lidi.info.unlp.edu.ar.

[*] *Esta investigación es parcialmente financiada porTelefónica de Argentina, CIC, CyTED y Fundación YPF.*

1. INTRODUCCIÓN

La utilización de arquitecturas como Clusters, Multiclusters y Grid, comunicadas vía mensajes y soportadas por redes de diferentes características y topologías se ha generalizado, tanto para el desarrollo de algoritmos paralelos como para el de servicios WEB distribuidos [1] [2].

Un *cluster* es una clase de sistema de procesamiento distribuido compuesto por un conjunto de máquinas *stand-alone* interconectadas trabajando cooperativamente como un recurso de cómputo único e integrado [3]. Al conectar dos o más clusters sobre una LAN o WAN se tiene alguna forma de *multicluster*.

Diferentes variantes de multicluster se obtienen si todos los clusters están sobre una misma red o enlazan diferentes redes; si el soporte de sistema operativo es común a todos los componentes; si cada cluster es homogéneo o heterogéneo; si la red de comunicaciones tiene un ancho de banda fijo entre nodos o es variable (típico de una WAN utilizando Internet) y si cada cluster está dedicado a la aplicación definida para el multicluster o la comparte con otras tareas.

Un *Grid* es un tipo de sistema paralelo y distribuido que permite compartir, seleccionar y agregar recursos autónomos distribuidos geográficamente tales como computadoras, software, datos, bases de datos, dispositivos especiales, instrumentos y personas. Esta configuración colaborativa depende de la disponibilidad, capacidad, costo y requerimientos del usuario [4]. También un Grid puede definirse como un entorno virtual de procesamiento de información donde el usuario tiene la "ilusión" de un único y potente recurso de cómputo que en realidad se encuentra distribuido [5].

Actualmente existen numerosas áreas de aplicación para estas arquitecturas, tales como: cómputo científico, simulación, modelos industriales, medicina, comercio electrónico, manejo de bases de datos distribuidas, Internet (portales, Web services), E-Government o aplicaciones críticas tales como reactores nucleares, bancos, armas militares o control industrial en tiempo real. [6].

Algunas características de un entorno Grid [7]:

- Los recursos y servicios pueden unirse o dejar el Grid dinámicamente.
- El Grid integra recursos (procesadores, instrumentos, bases de datos, etc) que son heterogéneos, geográficamente distribuidos y en general conectados por una WAN.
- Los recursos pueden ser accedidos *on-demand* por un conjunto de usuarios que configuran una comunidad virtual.
- El Grid está configurado usando protocolos e infraestructura de propósito general, no necesariamente común a todos sus nodos.
- Problemas clásicos de los sistemas distribuidos como la tolerancia a fallas, el balance de carga o la calidad de servicio aparecen con un grado mayor de complejidad en el Grid.

Si miramos las funcionalidades de la estructura de un Grid "por capas" podemos señalar las siguientes:

A. El nivel más bajo constituido por los servicios Grid ("Factory layer") para dar soporte a la utilización de los recursos locales (procesadores, datos, red).

B. La capa que da los servicios de autenticación y seguridad para permitir el intercambio de datos entre resursos remotos ("Connective Layer).

C. La capa de administración de recursos ("Resource Layer") que permite compartirlos y establecer una conexión lógica entre ellos.

D. La capa colectiva es la que coordina las interacciones entre múltiples recursos asociados a procesos distribuidos físicamente.

E. Por último la capa de aplicación es la que trata de resolver la interacción con el usuario, de modo que éste visualice en forma "transparente" la configuración virtual con la que trabajará.

Algunos autores consideran que un Grid es un "Cluster of Clusters", lo que resulta una definición algo restrictiva pero útil para la evolución de aplicaciones paralelas de Clusters a Grid. [8]

Pueden mencionarse algunas similitudes y diferencias:

- En un cluster normalmente se configura una única máquina paralela virtual que puede estar ejecutando una aplicación dedicada. Un Grid permite configurar múltiples máquinas paralelas virtuales para varios usuarios/aplicaciones simultáneas.
- Tanto clusters como Grids se basan en procesadores heterogéneos. Sin embargo en Grid esta heterogeneidad se extiende a la red de comunicaciones y al tipo de componentes en cada nodo que pueden ser procesadores, instrumentos, sensores, etc.
- El middleware necesario para Grid es más complejo que el de los clusters. Fundamentalmente, para configurar la máquina paralela virtual es necesario una etapa de identificación de recursos físicos y su ubicación. Además en el Grid es necesario monitorear la ejecución de tareas sobre múltiples máquinas virtuales con usuarios de diferente nivel y con distintos derechos de acceso a los recursos.
- Asimismo las herramientas para el desarrollo de aplicaciones requieren un mayor nivel de abstracción en Grid, por la complejidad y variedad de los múltiples usuarios que pueden utilizar la arquitectura.

Es interesante notar que una estructura de multicluster, visualizada como un *número limitado de clusters dedicados que cooperan en una única aplicación paralela*, es un punto intermedio entre clusters y Grid y requerirá algunos servicios especiales en su middleware (especialmente para autenticar derechos de usuarios que acceden a recursos remotos).

El desarrollo de algoritmos paralelos sobre arquitecturas débilmente acopladas y geográficamente distribuidas como multiclusters y Grid presenta nuevos desafíos, entre los que pueden mencionarse:

- La heterogeneidad de las comunicaciones y su costo variable según los nodos a conectar dificulta la asignación óptima de tareas a procesadores y el balance dinámico de la carga.
- Los modelos para predicción de performance son complejos y agregan la incertidumbre del ancho de banda efectivo en el caso de emplear Internet.
- La granularidad óptima a emplear depende de la relación entre potencia de cómputo local y remota. Muchas veces la configuración efectiva de los nodos remotos a utilizar no es conocida a priori.
- De mínima se requiere un soporte de middleware para la autenticación de los usuarios y de los recursos remotos a utilizar.
- El modelo cliente-servidor (paradigma muy empleado en algoritmos paralelos sobre clusters) se torna ineficiente al incrementar el número de nodos. Esto requiere la reformulación de algoritmos que ejecutan sobre clusters.
- Las herramientas de software más generalizadas en clusters (ej. MPI o PVM) tienen restricciones al tratar de emplearlas en topologías que conectan diferentes redes.

Estos elementos llevan al desarrollo de nuevos paradigmas, lenguajes y ambientes de software para el desarrollo de sistemas paralelos sobre multicluster y grid. Asimismo los recursos de middleware necesarios para la visión de "máquina paralela virtual" son más complejos y aún no se han estandarizado.

El desarrollo de algoritmos paralelos sobre clusters (y multiclusters dentro de la misma red) ha utilizado principalmente bibliotecas de comunicaciones como PVM y MPI [9] [10]. Sin embargo al tratar de ejecutar las mismas aplicaciones sobre un Grid surgen problemas con el manejo de estas bibliotecas de comunicaciones, que actualmente están buscando un nuevo standard para Grid. Esto se debe principalmente a la necesidad de resolver el acceso remoto en forma transparente para la operación de procesos y el acceso a los datos.

Básicamente el soporte de servicios del middleware de Grid debe proveer las siguientes funciones:

- Identificar los recursos disponibles y su ubicación física. Esto obliga a mantener información distribuida del estado de los recursos.

- Administrar datos distribuidos entre procesos/procesadores ubicados en diferentes nodos del Grid.

- Manejar interacciones (comunicación, sincronización) entre procesos que pueden estar geográficamente distribuidos.

- Administrar algún mecanismo de seguridad común, transparente para los usuarios finales.

- Controlar la ejecución de "sesiones" y "programas" independientemente de su origen y del equipamiento distribuido asociado.

El análisis por capas que se ha realizado previamente del software de base para administrar un Grid hace evidente que hay un overhead natural al pasar una aplicación que se ejecuta sobre un cluster local o sobre un multicluster dedicado con usuarios autorizados en todos los nodos a una estructura de Grid geográficamente distribuida, que integra diferentes redes locales con alguna forma de red WAN [11] [12].

En este trabajo se enfoca el problema de la migración de cluster/multicluster a Grid en el caso de algoritmos con distribución de carga estática y de alto porcentaje de procesamiento en relación con las comunicaciones.

Se ha elegido este tipo de problema "simple" para poder identificar claramente el overhead "básico" que genera el middleware de Grid. En particular se eligió el caso de N-Reinas porque había sido extensamente estudiado previamente por los autores sobre cluster y multicluster, lo que permitía trabajar sobre un código probado y con resultados conocidos [13] [14] [15].

Se configuró un GRID experimental, con todos los componentes del middleware conectando 2 clusters de 8 máquinas cada uno (de hecho los resultados se pueden extender fácilmente a más clusters y/o máquinas) y se estudió el overhead, sin modificar el código de base.

2. ANALISIS DEL PROBLEMA DE LAS N-REINAS

El problema de las N-reinas consiste en ubicar N reinas en un tablero de NxN de tal manera que ninguna de ellas ataque a otra [16][17]. Una reina ataca a otra si se encuentran en la misma diagonal, fila o columna.

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 [18] [19].

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 1.a). Para determinar la cantidad de soluciones a partir de la fila i (tal que toda fila j con $j \leq i$ 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 1.b). 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 1.c).

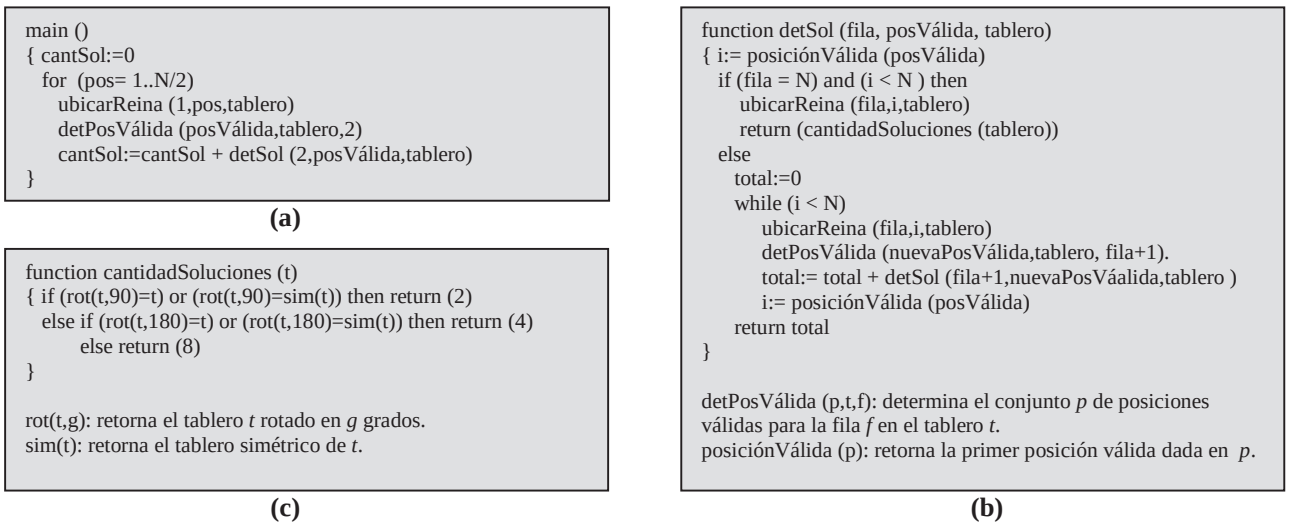


Figura 1. Pseudocódigo de la solución secuencial

Solución paralela

Para la solución paralela de este problema, se ubica la reina en 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 [13] [14]. Para esto se usan las cuatro primeras filas para formar cada una de las combinaciones a resolver. De esta manera se obtienen N^4 combinaciones diferentes para distribuir entre todos los procesadores heterogéneos, siendo N el tamaño del tablero.

Se ha elegido trabajar con el paradigma Master-Slave. La Figura 2.a muestra el pseudocódigo del algoritmo utilizado por el proceso Master, el cual reparte todas las combinaciones entre los procesos Slaves al comenzar la aplicación (distribución estática), resuelve un subconjunto de ellas, y luego junta los resultados de todos los slaves. A su vez la Figura 2.b muestra el pseudocódigo de los procesos Slaves los cuales calculan las posibles soluciones al problema a partir de las combinaciones que le son asignadas, y por último la Figura 2.c muestra el pseudocódigo de un módulo utilizado para determinar las soluciones a partir de una cierta configuración inicial.

En esta clase de problemas el tiempo de comunicación entre procesos *Tc* no es significativo frente al tiempo de procesamiento local *Tp* (*Tp >> Tc*). Esta característica permite identificar el overhead generado por el middleware utilizado para GRID sin que las comunicaciones influyan en este análisis.

En este punto vale aclarar que en trabajos previos se han realizado algoritmos más eficientes al utilizar distribuciones dinámicas o semidinámicas que logran un mejor balance de carga a costa de un aumento en la cantidad de comunicaciones [15].

```
main proceso 0 ()
{ //Envia a cada procesador las combinaciones que debe evaluar
  scatter (rangos,0)
  //Procesa su parte
  cantSol:=0
  while  (procesador 0 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)
    determina la ubicación en la fila 4 (p4)
    cantSol:= cantSol + detSolParcial (p1,p2,p3,p4, tablero)
  //Recibe los resultados de todas las tareas
  reduce(cantSol, ADD, total,0)
 }
```

(a)

```
main procesador i ()   // Con i > 0
{ //Recibe las combinaciones a realizar
  scatter (rangos,0)
  //Procesa su parte
  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)
      determina la ubicación en la fila 4 (p4)
      cantSol:= cantSol + detSolParcial (p1,p2,p3,p4,tablero)
  //Envia los resultados de todas las tareas
  reduce(cantSol, ADD, total,0)
 }
```

(b)

```
function detSolParcial (posFila1, posFila2, posFila3, posFila4, tablero)
{ ubicarReina (1,posFila1,tablero)
  ubicarReina (2,posFila2,tablero)
  ubicarReina (3,posFila3,tablero)
  ubicarReina (4,posFila4,tablero)
  detPosVálida (posVálida, tablero,5).
  return detSol  (5, posVálida, tablero)
}

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

(c)

Figura 2. Pseudocódigo de la solución paralela. La función detSol es la descripta en la Figura 1.b.

3. CONFIGURACION EXPERIMENTAL UTILIZADA

Se configuró una GRID utilizando dos clusters (“Foster Cluster” y “Newton Cluster”) del Instituto, añadiéndole a cada cluster un nodo cabecera (*head node)* como nodo GRID utilizando Globus Toolkit 4.0.4 como GRID Middleware. Ademas se agregaron servidores DNS, Web (*GRID Portal)* y un Router. La Figura 3 muestra gráficamente la arquitectura configurada.

Los clusters utilizados poseen las siguientes características técnicas:

Foster Cluster	***Newton Cluster***
8 Nodos PC [Foster1..Foster8] Intel Pentium III 700 MHz 256 MB RAM HD 20 GB Fast Ethernet 100 Mbps LAM-MPI 7.1.3	*8 Nodos PC [Newton1..Newton8]* Intel Pentium 4 2,66 GHz 512 MB RAM HD 40 GB Fast Ethernet 100 Mbps LAM-MPI 7.1.3
Head Node [Foster] Intel Pentium 4 2,66 GHz 512 MB RAM HD 80 GB Fast Ethernet 100 Mbps LAM-MPI 7.1.3	*Head Node [Newton]* Intel Pentium 4 2,66 GHz 512 MB RAM HD 80 GB Fast Ethernet 100 Mbps LAM-MPI 7.1.3

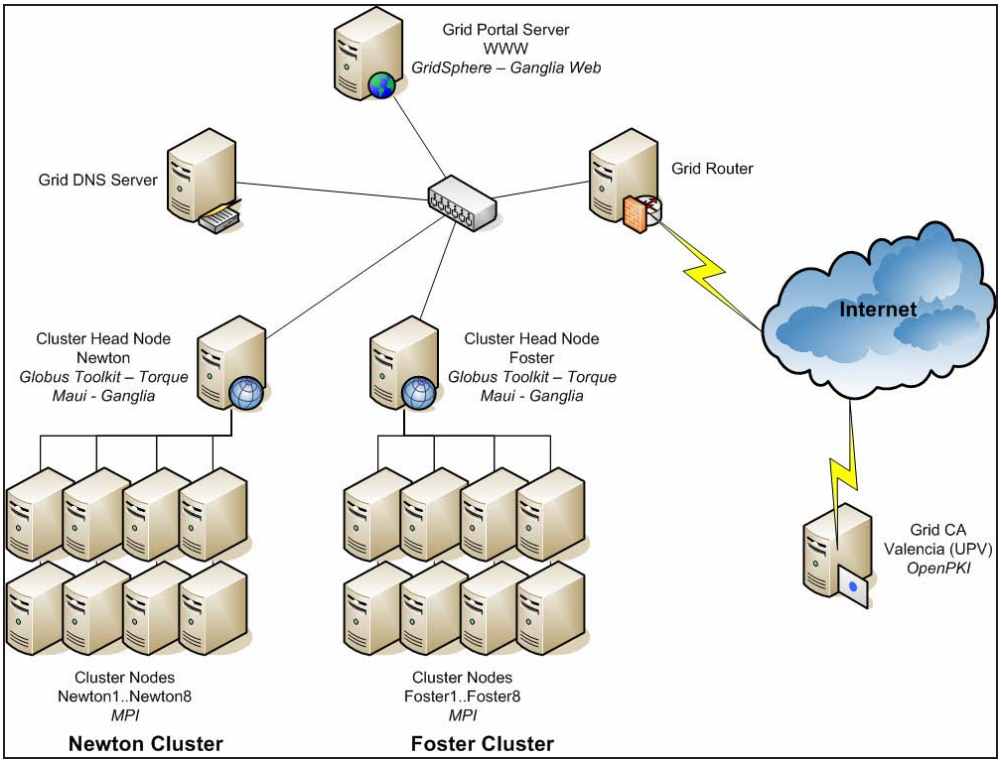


Figura 3. Infraestructura y topología de la III-LIDI GRID.

En ambos head nodes se utilizó Globus Toolkit 4.0.4 como Grid Middleware utilizando todos sus componentes (GridFTP, RFT, GRAM, WS-GRAM, RLS, MDS).

Se utilizó LAM-MPI 7.1.3 como librería de pasaje de mensajes en ambos clusters. Además, se utilizó Torque Resource Manager como manejador de recursos y Maui Cluster Scheduler. Estos servicios administran los recursos y solicitudes de procesamientos sobre los clusters.

Se utilizó la entidad certificante (CA) del proyecto CYTED, ubicada físicamente en la Universidad Politécnica de Valencia (UPV).

Para integrar la infraestructura multicluster al GRID, se configuró WS-GRAM interactuando con Torque logrando así el acceso a los recursos de los cluster a través del GRID.

Como herramienta de monitoreo del GRID se configuró Ganglia y su front-end web. Además, se instaló el Portal Web GridSphere para facilitar el uso del GRID a los usuarios.

La secuencia de ejecución sobre la infraestructura GRID es la siguiente:

- Se tiene un programa paralelo implementado en lenguaje C, empleando la biblioteca MPI.
- Se divide la ejecución del programa entre los dos *head nodes*, Foster y Newton.
- Cada nodo recibe el pedido de ejecución de su parte en paralelo a través de WS-GRAM.
- En cada *head node*, WS-GRAM interactúa con el Resource Manager “Torque” y el Cluster Scheduler “Maui” para procesar la ejecución.

- El Resource Manager de cada nodo interactúa a su vez con el ambiente MPI ejecutando el programa sobre el cluster.
- Cuando la ejecución sobre el cluster en Foster finaliza, éste envía a través de GridFTP los resultados a Newton.
- Cuando Newton finaliza la ejecución, lee los resultados enviados por Foster y corre un tercer programa que realiza la unión de los resultados de ambos clusters.

4. RESULTADOS OBTENIDOS

La Tabla I muestra los tiempos (en segundos) requeridos para la ejecución secuencial, paralela sobre la arquitectura de cluster y paralela sobre una arquitectura GRID para tableros de tamaño 17x17, 18x18, 19x19 y 20x20.

Tamaño	Tiempo Secuencial	Tiempo utilizando Cluster	Tiempo utilizando GRID
17	51,63	19,25	41,32
18	385,25	136,18	155,62
19	3025,65	975,77	997,21
20	24478,21	8547,53	8570,07

Tabla I. Tiempos secuencial y paralelo (Cluster y GRID) de las pruebas realizadas.

La Figura 4 muestra el overhead en segundos resultante del empleo del middleware de Grid para los diferentes tamaños de tablero. Se puede observar que el overhead prácticamente no depende del trabajo realizado (que crece exponencialmente con el tamaño de los tableros).

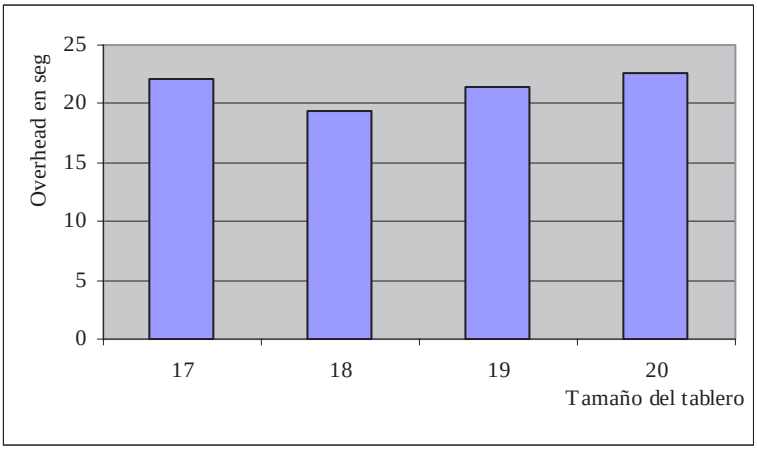


Figura 4. Overhead producido por el middleware de GRID.

Para analizar el rendimiento del algoritmo en la arquitectura paralela se utiliza la métrica speedup dada por la fórmula:

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

En el caso de una arquitectura heterogénea el "Tiempo Secuencial" está dado por el tiempo del algoritmo secuencial ejecutado en la máquina con mayor potencia de cálculo.

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.

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

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:

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

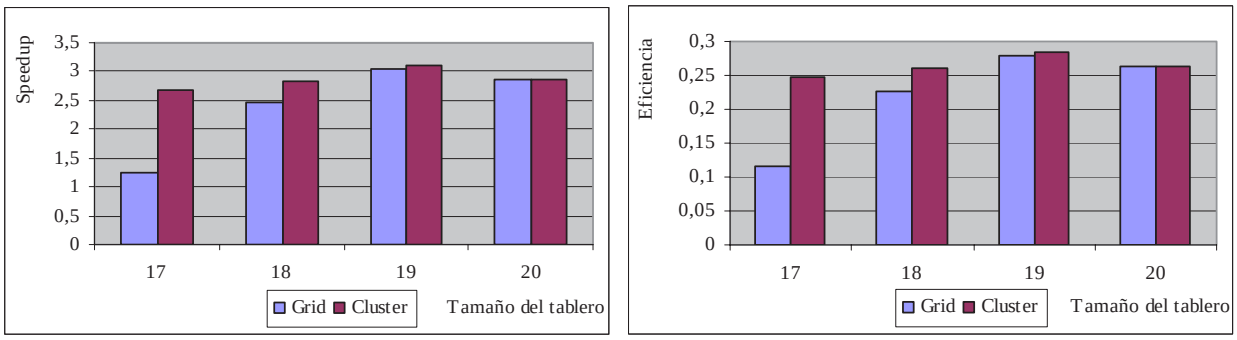
Como se dijo anteriormente la arquitectura utilizada está formada por 2 clusters de 8 máquinas cada una donde los procesadores que componen uno de los clusters poseen un 36% de la potencia de cálculo de los que pertenecen al otro cluster. Por lo tanto el speedup teórico de la arquitectura es 10,88 (0,36*8 + 1*8).

La Tabla II muestra el speedup y la eficiencia obtenidos en cada una de las arquitecturas para los diferentes tamaños de tablero.

	Grid		**Cluster**	
Tamaño	**Speedup**	**Eficiencia**	**Speedup**	**Eficiencia**
17	1,249	0,114	2,681	0,246
18	2,475	0,227	2,828	0,260
19	3,034	0,278	3,100	0,284
20	2,856	0,262	2,863	0,263

Tabla II. Speedup y eficiencia de las pruebas realizadas

Por último en las Figuras 5.a y 5.b se visualizan los valores mostrados en la Tabla II.



(a) **(b)**

Figura 5. (a) Speedup y (b) Eficiencia de las pruebas realizadas en Cluster y GRID.

5. CONCLUSIONES Y LINEAS DE TRABAJO FUTURO

El resultado más importante es que para esta clase de aplicación (de alto porcentaje de tiempo de procesamiento respecto del tiempo de comunicación) el overhead que introduce el software de Grid representa prácticamente un tiempo constante, que no cambia al escalar el problema (de N=17 a N=20).

Otro aspecto interesante es que la migración de la aplicación es prácticamente transparente, manteniendo el código original y agregando los servicios de autenticación, ejecución remota y comunicación inter-cluster a través del middleware de Grid. Los resultados obtenidos respecto del overhead básicamente no dependen del número de clusters que se conecten en el Grid, ni tampoco del número de máquinas de cada uno de ellos.

Actualmente se está trabajando en analizar problemas con mayor comunicación y con alguna forma de balanceo dinámico en la ejecución (lo que obliga a una interacción durante la ejecución entre procesos en diferentes nodos del Grid). Asimismo se estudia la eficiencia alcanzable al escalar el problema y la dimensión de los clusters.

REFERENCIAS

[1] Z. Juhasz (Editor), P. Kacsuk (Editor), D. Kranzlmuller (Editor). “Distributed and Parallel Systems: Cluster and Grid Computing”. The International Series in Engineering and Computer Science. Springer; 1 edition. 2004.

[2] Di Stefano M. “Distributed data management for Grid Computing”. John Wiley & Sons Inc. 2005.

[3] Grid Computing and Distributed Systems (GRIDS) Laboratory - Department of Computer Science and Software Engineering (University of Melbourne). “Cluster and Grid Computing”. 2007. http://www.cs.mu.oz.au/678/.

[4] Joseph J., Fellenstein C. “Grid Computing”. On Demand Series. IBM Press. 2003.

[5] Foster I., Kesselman C., Kaufmann M. “The Grid 2: Blueprint for a New Computing Infrastructure”. The Morgan Kaufmann Series in Computer Architecture and Design. 2 edition. 2003.

[6] “CSM23 Grid Computing”. http://www.computing.surrey.ac.uk/courses/csm23.

[7] Berman F., Fox G., Hey A. “Grid Computing: Making The Global Infrastructure a Reality”. John Wiley & Sons. 2003.

[8] De Giusti A. et al. “Parallel algorithms on Multi-Cluster Architectures using GRID Middleware. Experiences in Argentine Universities”. Proceedings of the I Iberian Grid Infrastructure Conference. Spain. 2007. Pag. 322-332

[9] “PVM Parallel Virtual Machine”. http://www.csm.ornl.gov/pvm/

[10] “MPI Message Passing Interface”. http://www.mpi-forum.org/

[11] “Grid Computing Info Centre”. http://www.gridcomputing.com/

[12] “The Globus Alliance”. http://www.globus.org/

[13] De Giusti L., Novarini P., Naiouf M. R., De Giusti A. E. “Parallelization of the N-queens problem. Load unbalance analysis”. Workshop de Procesamiento Paralelo y Distribuido. 2003. pag.397-403

[14] De Giusti Laura, Naiouf Marcelo, Chichizola Franco, De Giusti Armando. Informe Técnico III-LIDI “Balance de Carga en procesamiento Multicluster. Aplicación al problema de las N-Reinas”. 2005.

[15] Naiouf M. R., De Giusti L. C., Chichizola F., De Giusti A. E. “Dynamic Load Balancing on Non-homogeneous Clusters”. G.Min et al. (Eds.): ISPA 2006 Ws, LNCS 4331, pags. 65-73. Springer – Verlag. Berlin Heidelber 2006.

[16] Bruen A, Dixon R. “Then n-queens problem. Discrete mathematics”. 12:393-395. 1997.

[17] Hedetniemi S, Hedetniemi T, Reynolds R. “Combinatorial problems on chessboards: II”. Chapter 6 in domination in graphs: advanced topic, pág 133-162. 1998.

[18] Somers J. “The N-queens problem a study in optimization”. www.jsomers.com/nqueen_demo /nqueens.html.

[19] Takaken. “N-queens problem (number of solutions)”. http://www.ic-net.or.jp/home/takaken/e/queen/.