Universidad de Los Andes
Facultad de Ingeniería
Postgrado de Computación

Sistema Manejador de Ambientes Reconfigurables para Procesamiento Paralelo/Distribuido

Tesis de Grado presentada por:
Ing. Francisco J. Hidrobo T.
para optar al título de:

Magister Scientiae en Computación

Noviembre, 1998

Mérida, Venezuela.

Resumen

En este trabajo, se presenta el diseño de un Sistema Manejador de Ambientes Reconfigurables para procesamiento Paralelo/Distribuido. Así, se presentan varios enfoques de diseño, uno en el que el programa se adapta al sistema y otro que sigue el procedimiento inverso. Los enfoques son basados en la teoría de grafos, por lo que se presentan varios problemas de tipo NP-completos a ser resueltos en la implementación del sistema. Un primer problema es el de particionamiento de grafos (agrupamiento de la tareas) y el segundo problema es la determinación del máximo D-acoplamiento del grafo de agrupamiento. Se usan los Algoritmos Genéticos como técnica para encontrar buenas soluciones a dichos problemas.

Además, la propuesta abarca un diseño integral e integrado de diversas entidades que forman parte del sistema; para esto se recurre a la teoría de sistemas multiagentes que posean arquitecturas adaptables a las características de la plataforma.

Índice General

Índice de Figuras

Índice de Tablas

Introducción

La utilización de máquinas masivamente paralelas en el campo científico se ha incrementado aceleradamente en los últimos años. Muchas aplicaciones secuenciales de altos requerimientos computacionales están siendo migradas a plataformas paralelas. Sin embargo, es posible observar que estos procesos de migración o de construcción de programas paralelos refleja en muchos casos características particulares de la plataforma de comunicación en la cual se deberían ejecutar los programas (plataforma ideal de ejecución); es decir, las aplicaciones están atadas a una topología de comunicación. En este sentido, se necesitaría una amplia variedad de topologías para dar servicio a todas las posibles aplicaciones.

Por otro lado, el rendimiento de un programa paralelo depende en gran medida de la configuración de la arquitectura sobre la cual se este ejecutando. El desarrollo de programas paralelos con la restricción de tener que adaptarse a una topología comunicacional fija es un procedimiento difícil y poco natural. La búsqueda del mejor algoritmo para resolver un problema dado, no debe tener restricciones topológicas. Una arquitectura en la cual la red de interconexión pueda ser modificada dinámicamente, en función de las necesidades de comunicaciones de los programas, permite remediar estas limitaciones. Esto plantea la necesidad de disponer de plataformas reconfigurables que den un mayor grado de flexibilidad y disponibilidad.

Una arquitectura paralela es *reconfigurable* si la topología de su red de interconexión puede cambiar en función de los programas que van a ejecutarse sobre ella, o bien, pueden realizarse transformaciones en los programas para que se adapten a la topología sin perdidas significativas en el rendimiento. Las arquitecturas paralelas reconfigurables han sido usadas en multiprocesadores a memoria compartida. Estos últimos años han empezado a ser usados en arquitecturas a memoria distribuida. Por consiguiente, son muchos los esfuerzos que se están realizando en las áreas de arquitectura de computadores y sistemas operativos dirigidos, principalmente, a hacer mas eficientes estas plataformas. No obstante, los problemas asociados a la adaptación de las aplicaciones a las plataformas están lejos de ser completamente resueltos.

Los Sistemas Multiagentes están siendo ampliamente estudiados en la actualidad puesto que constituyen el punto de enlaces de diversas áreas, tales como: la inteligencia artificial distribuida, computación emergente y la vida artificial, entre otras. Existen muchos trabajos orientados a la conceptualización de los sistemas multiagentes [1, 2, 3, 4, 5].

El enfocar una plataforma de procesamiento paralelo/distribuido como un sistema multiagentes podría conducir a un modelo unificador de dichas plataformas a través de un esquema de representación que este inmerso en el propio sistema; de esta manera, el conocimiento de las características de la plataforma se incluyen en el sistema multiagentes que la maneja.

Este trabajo apunta hacia la conceptualización y el modelado de de las plataformas de procesamiento paralelo/distribuido, así como del análisis y solución de los problemas asociados a la adaptación de aplicaciones a dichas plataformas. En este sentido, es primordial el conocimiento y la caracterización de tales problemas, así como de los elementos teóricos y de las técnicas que serán utilizadas para el planteamiento tanto a nivel específico como global de la solución. Entre los trabajos más cercanos al que se propone están los realizados en el grupo de agentes autónomos del MIT [6], en el cual se presenta el diseño de un sistema multiagentes, denominado *Challenger*, para la asignación distribuida de recursos (CPU). El sistema esta formado por un conjunto de agentes que manejan los recursos locales (tiempo de CPU) y se comunican con los otros para compartirlos.

Este documento está organizado de la siguiente manera: el capítulo 1 presenta los aspectos relacionados con los ambientes de procesamiento paralelo/distribuido y los problemas asociados a tales ambientes; además, incluye elementos de la teoría de grafos y algunos de los problemas clásicos de esta área que se estudian en este trabajo. Posteriormente, el capítulo 2 describe los conceptos básicos de las técnicas de computación inteligente. En el capítulo 3 se presenta la propuesta del sistema manejador del ambiente, su funcionamiento y las alternativas de planificación. Luego, en el capítulo 4, se plantea el sistema manejador del ambiente basado en un enfoque multiagentes en el cual algunos elementos son enfocados como agentes con actividades y relaciones que conllevan a la solución global del problema. Después, el capítulo 5 describe y muestra las pruebas realizadas y presenta un análisis de los resultados. Por último, se presentan las conclusiones.

Capítulo 1

Marco Teórico

La complejidad y diversidad de los problemas asociados al diseño de un sistema reconfigurable de procesamiento paralelo hacen necesario un análisis y estudio de dichos problemas. La caracterización y representación son los aspectos fundamentales en tal análisis.

En este capítulo se presentan y discuten los elementos que forman parte de una plataforma de procesamiento paralelo, su caracterización y los problemas que deben ser resueltos en tales ambientes para lograr un rendimiento óptimo. Además, se hace una introducción teórica a los grafos como herramienta de modelado. Después, se describen los problemas combinatorios de partición de grafos, circuito Hamiltoniano máximo, y D-acoplamiento, presentes en la teoría de grafos, los cuales tienen una gran relevancia en el sistema manejador que se propone.

1.1 Reconfiguración de Ambientes de Procesamiento Paralelo/Distribuido

Cuando se habla de arquitecturas paralelas reconfigurables, se hace referencia a aquellos sistemas donde la red de interconexión no es fija, sino que es definida en función del programa que se va a ejecutar. Es decir, la reconfigurabilidad de ambientes de procesamiento paralelo se refiere básicamente a la posibilidad de definir la topología de interconexión de los procesadores para cada aplicación. Dicha definición se refiere al número de procesadores y los enlaces entre ellos; de esta manera quedará identificada una topología particular.

Es posible que la definición hecha pueda realizarse de forma real, es decir, los procesadores se interconectan físicamente como lo indica la topología especificada; en este caso se estaría hablando de *reconfiguración por hardware o reconfiguración por hardware programable (switching)*. Alternativamente, la definición topológica puede hacerse completamente por software, implementandose a través de una *topología virtual*. En este caso, la creación y la manipulación de dicha topología se hace por *software*.

Bajo el esquema de topología virtual, la interconexión real de los procesadores sólo

es conocida por el **Sistema Manejador del Ambiente** (SMA). El SMA debe determinar la topología sobre la cual se debe ejecutar una aplicación dada, para generarla virtualmente. Además, realizará todas las tareas de gestión sobre esa topología, de manera de optimizar el tiempo de ejecución de la aplicación.

Si la reconfiguración se realiza antes de iniciarse la ejecución del programa, se esta hablando de una *reconfiguración estática*, de lo contrario, si ésta es modificable en el curso de la ejecución de un programa y/o de la operación del sistema, se habla de una *reconfiguración dinámica.*

1.1.1 Caracterización de los Ambientes de Procesamiento Paralelo/Distribuido

Un ambiente de procesamiento Paralelo/Distribuido (APPD) puede definirse como una plataforma compuesta de diversos recursos (Procesadores, Sistemas de Comunicación, Discos, Memoria, Sistema Operativo, etc.) que proveen servicios computacionales a aplicaciones que resuelven problemas específicos. La descripción de un APPD se hace en término de los componentes del mismo y de la interrelación entre éstos. A continuación, se presentan algunos de los componentes de un APPD:

- Unidades de procesamiento (*CPU's*)
  Responsable de las tareas de control y ejecución de los procesos. Caracterizadas, principalmente, por la capacidad de memoria y la velocidad relativa (la relación entre la velocidad del CPU y la velocidad de algún CPU que sea tomado como referencia para todos).

- Sistema de interconexión
  Representa los mecanismos, la topología y la tecnología usada para la comunicación entre las unidades de procesamiento.

- Aplicaciones de gestión (Ejemplo: Asignación de tareas, Balance de cargas , etc.)
  Estas permiten definir los esquemas/políticas utilizadas para administrar los recursos en el sistema, con el fin de optimizar el rendimiento.

- Estructura de las aplicaciones (paradigma de programación paralela)
  Expresa el flujo de ejecución y las características comunicacionales de las aplicaciones. Normalmente, dicha estructura es representada usando un grafo de tareas, con la respectiva descripción de cada tarea y los requerimientos de comunicación entre las mismas.

1.1.2 Problemas ligados a la reconfiguración

Nociones tales como *la asignación de tareas, el equilibrio de carga, la descomposición de programas, la tolerancia a fallas*, deben ser tomados en cuenta al momento de reconfigurar una máquina paralela. En esta sección, se hará una breve introducción de estos aspectos.

Tolerancia a fallas

Un sistema falla cuando alguno de sus componentes tiene un problema (mal funcionamiento, etc.). En ciertos sistemas, una falla puede ser catastrófica; así, hacerlos tolerantes a fallas es fundamental. Existen diferentes nociones que deben ser usadas para hacer un sistema tolerante a fallas, tales como *la migración de procesos*, *la reconfiguración de máquinas*, etc. En este último caso, lo que se busca es tomar en cuenta la nueva topología de interconexión después que una falla a ocurrido, sea a nivel de los procesadores o de los canales de comunicación, para que los programas puedan continuar su ejecución.

Descomposición de programas y Multiprogramación

Las máquinas paralelas permiten, en principio, que un programa pueda ser dividido en tareas las cuales pueden ejecutarse simultáneamente. Sin embargo, es probable que el número de tareas obtenidas sea mayor que el número de procesadores disponibles; por lo cual algunas tareas deben esperar por algún procesador o "competir" por este. Esto último es conocido como *multiprogramación*. Así, cuando se desea ejecutar varias tareas en una misma máquina puede emplearse la multiprogramación, a sabiendas de que esto podría degradar significativamente el rendimiento de cada procesador y/o programas por la interferencia en el procesamiento de tareas que deberían ser ejecutadas en paralelo.

La multiprogramación es fundamental bajo el esquema de ambientes reconfigurables con topología virtual; puesto que un procesador puede formar parte de dos topologías distintas o representar a dos nodos distintos de una misma topología.

Asignación de Tareas

En todo ambiente de reconfiguración de máquinas paralelas, existe un proceso a través del cual se deben asignar un conjunto de tareas al conjunto de procesadores del sistema. Este proceso de asignación debe responder a preguntas, tales como: ¿Cuales tareas serán asignadas a que procesadores?, ¿Cuales tareas serán agrupadas para ser ejecutadas en un mismo sitio?, etc. Para realizar este proceso, se pueden usar los siguientes criterios de optimización:

- Minimización de la comunicación entre tareas comunicantes.
- Repartición equitativa de la carga de trabajo entre los diferentes procesadores del sistema.
- Minimización de los tiempos de ejecución de las tareas.

En el caso de la asignación de tareas, se puede hablar de una *asignación estática* cuando esta es realizada antes de la ejecución del programa, sin que cambie durante su ejecución; y de una *asignación dinámica*, la cual consiste en asignar las tareas de

un programa durante su ejecución. En este caso, las nociones de *migración de tareas y equilibrio dinámico de la carga*, son usados para realizar este proceso.

Equilibrio de la carga

Equilibrio de la carga es una de las alternativas al distribuir la carga total en un sistema (la otra es la repartición de la carga). En este caso, lo que se busca es mantener una carga total equilibrada entre los diferentes recursos del sistema, tomando en consideración las características particulares de cada procesador, la aplicación, el sistema de comunicación, etc. La Figura 1.1 muestra la situación que puede presentarse por desequilibrio de carga; en este caso, existen algunos nodos que tienen un nivel de ocupación mucho mayor que otros. Cuando se desea optimizar el rendimiento de un APPD, se debe asegurar que la carga esta uniformemente repartida (procesadores homogéneos), ya que de esta manera todos los procesadores tendrán un uso similar y no habrá procesos compitiendo por un procesador mientras haya otros libres.

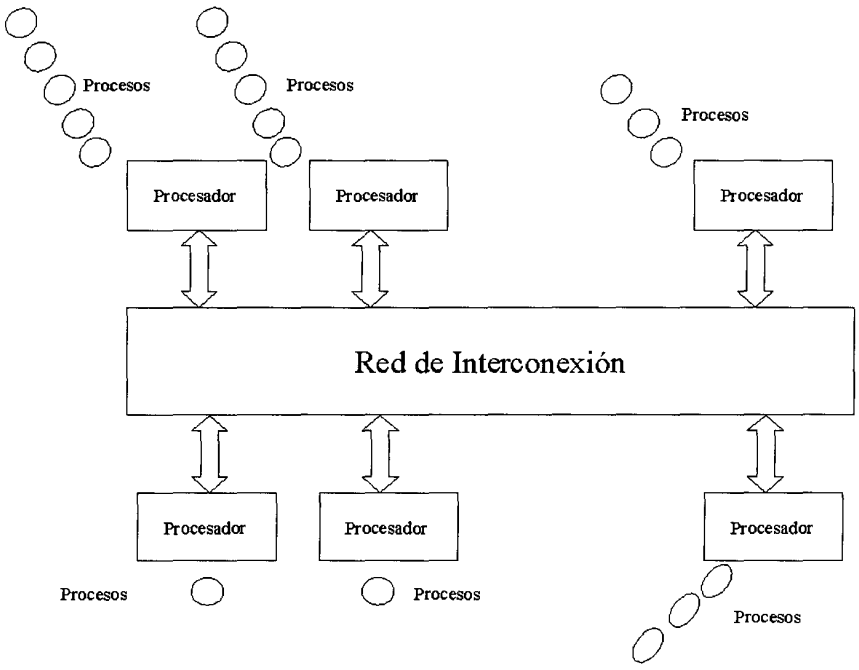


Figura 1.1: Ambiente de procesamiento sin equilibrio de carga

Migración de tareas

La migración de tareas se refiere al movimiento de los procesos entre procesadores sin que estos procesos hayan sido finalizados. Esta migración se realiza, principalmente,

para asegurar el equilibrio de la carga o para garantizar la culminación de las tareas (por falla en algún procesador). En el proceso de migración intervienen una serie de aspectos como la transferencia (¿cuándo?), la selección (¿cuál tarea?), y la localización (¿con quién intercambia?).

1.1.3 Sistema Manejador del Ambiente (SMA)

El SMA es el *ente* que sirve como mediador entre la plataforma paralela y las aplicaciones del usuario. Específicamente, el SMA debe optimizar el uso de los recursos (CPUs) minimizando el tiempo de ejecución de múltiples aplicaciones con diversos requerimientos. Desde el punto de vista global, el SMA debe considerar:

- *Cambios en la Estructura del Sistema:* son derivados por modificaciones en los recursos que posee el sistema, debido a fallas o mantenimiento de estos, nuevos recursos en el sistema, etc.
- *Optimización del Uso de los Recursos:* de esta manera se responde a necesidades tales como equilibrar la carga en el sistema, minimizar los tiempos de ejecución de los programas, etc.
- *Cambios dinámicos en el Código de Ejecución:* esto puede generar cambios en la jerarquía de ejecución de los procesos de un programa (implicando, posiblemente, un cambio en la topología de comunicación) y en la carga de trabajo en el sistema.
- *Cambios debido a la Multiprogramación:* Esto ocurre cuando llegan nuevos programas al sistema o cuando culminan viejos programas, lo que implica demanda-/liberación de recursos, aumento/disminución de la carga de trabajo en el sistema, etc.

1.2 Teoría de Grafos

Un grafo, $G = (V, E)$, es un objeto matemático conformado por un conjunto de *vertices* ($V = v_1, v_2..., v_n$) y otro conjunto cuyos elementos se denominan arcos ($E = e_1, e_2, ...$), tal que cada arco e_k esta definido por un par no ordenado (v_i, v_j) de vertices [7].

La definición anterior permite que un arco este asociado con un par de vertices (v_i,v_i), recibiendo la denominación de *lazo*. Además, es posible que una par de arcos e_k y e_m tengan asociados el mismo par de vertices v_i,v_j, dichos arcos se llaman *arcos paralelos*. Así, un grafo simple esta definido como un grafo que no contiene ni lazos ni arcos paralelos. La forma gráfica, mas común, de representar un grafo es mediante un diagrama, en el cual los vertices se representan como puntos y los arcos por segmentos de líneas que unen dichos puntos.

Debido a la inherente simplicidad de los grafos, la teoría de grafos tiene un amplio rango de aplicaciones en ingeniería, física, ciencias sociales y biológicas, y otras áreas.

Un grafo puede ser usado para representar casi cualquier situación física que involucre objetos discretos y las relaciones entre los mismos.

A continuación, se presentan algunas definiciones relativas a los grafos que son de interés para este trabajo:

- **Grado**. Un concepto relacionado que debe definirse previamente es el de incidencia; cuando un vertice v_i es uno de los vertices de un arco e_k, v_i y e_k se dice que son incidentes uno con el otro. De esta manera, el grado, $d(v_i)$, de un vertice es el número de arcos que inciden en éste (Los lazos se cuentan dos veces).

- **Vertices vecinos**. Dos vertices, v_i y v_j, son vecinos si existe algún arco e_k que incida en ambos.

- **Camino**. Dados dos vertices u y v, un camino de u a v es una secuencia finita de vertices vecinos y arcos de G, representados como:
$$v_0 e_1 v_1 e_2 .... v_{n-1} e_n v_n$$
donde cada v_i representa un vertice y cada e_i representa un arco, $v_0 = u$ (punto de partida) y $v_n = v$ (punto final), para todo $i = 0, 2, ...n$.

- **Ruta**. Un ruta de u a v es un camino de u a v que no contiene arcos repetidos. Una ruta se expresa de la forma:
$$u = v_0 e_1 v_1 e_2 .... v_{n-1} e_n v_n = v$$
donde todos los e_i son distintos ($e_i \neq e_k$ para todo $i \neq k$).

- **Ruta simple**. Una ruta que no contiene vertices repetidos se denomina una ruta simple.

- **Camino cerrado**. Es un camino que comienza y termina en el mismo vertice.

- **Circuito**. Un camino cerrado que no contiene arcos repetidos es llamado un circuito.

- **Circuito Simple**. Es un circuito que no contiene vertices repetidos, excepto el primero y el último.

- **Circuito Hamiltoniano**. Un circuito simple que contiene todos los vertices de G se denomina un circuito hamiltoniano.

- **Grafo conexo**. Un grafo G es conexo, si y solo si, dados cualquier par de vertices u y v de G, existe un camino de u a v.

- **Grafos isomórficos**. Dos grafos, G y G', son isomórficos, si existe una correspondencia uno a uno entre sus vertices y sus arcos.

- **Subgrafos**. Un grafo g es subgrafo de un grafo G si todos los arcos y todos los vertices de g están en G, y cada uno de los arcos de g tiene los mismos vertices que en G.

La teoría de grafos ha representado un esquema de modelado bastante efectivo en algunos problemas de optimización combinatoria. Dichos problemas se caracterizan por la existencia de soluciones que se componen secuencialmente, es decir, dado un conjunto de elementos se pueden obtener diferentes arreglos ordenados de estos, permitiendo una basta cantidad de posibilidades [8]. El entendimiento de los problemas de optimización combinatoria es bastante sencillo, más no así su solución debido a que algunos de ellos son problemas de tipo NP-completos. Los problemas NP-completos se caracterizan por tener dos propiedades, la primera de ellas es que su complejidad es NO polinómica (el tiempo de ejecución crece exponencialmente con el tamaño del problema), y la segunda es el hecho de que el problema puede ser transformado en otro problema NP-completo [9].

En la siguientes secciones se decribirán, detalladamente, los problemas de optimización combinatoria presentes en el sistema.

1.2.1 Partición de Grafos

El problema de partición de grafos es un problema común en el área de computación paralela; puesto que describe la situación que se presenta cuando se requiere asignar N tareas de una aplicación paralela sobre K procesadores, manteniendo las restricciones de comunicación de dichas tareas. Esta sección presenta la descripción detallada del problema de particionamiento de un grafo.

Descripción

Este problema consiste en dividir un grafo G en varios subgrafos, tal que se minimice una función objetivo. Una posible definición es la siguiente:

Dado un grafo G, lo que se desea es particionar sus vertices en K subgrafos de tal manera que la suma de los pesos de los vertices por subgrafo sea igual (de ser posible) y la suma de los pesos de los arcos conformados por vertices que estén en diferentes subgrafos sea mínimo.

Para ilustrar el problema, en la Figura 1.2 se presenta un grafo de 13 vertices que ha sido particionado en 6 subgrafos.

Matemáticamente, el problema de partición de grafo puede ser definido como:

$$G = (\ V,\ A),$$

donde:

$V = \{1, ..., n\}$ es el conjunto de n vertices, con un arreglo de pesos asociado $P = \{p_1, p_2..., p_n\}$ donde cada p_i corresponde al peso del vertice i.

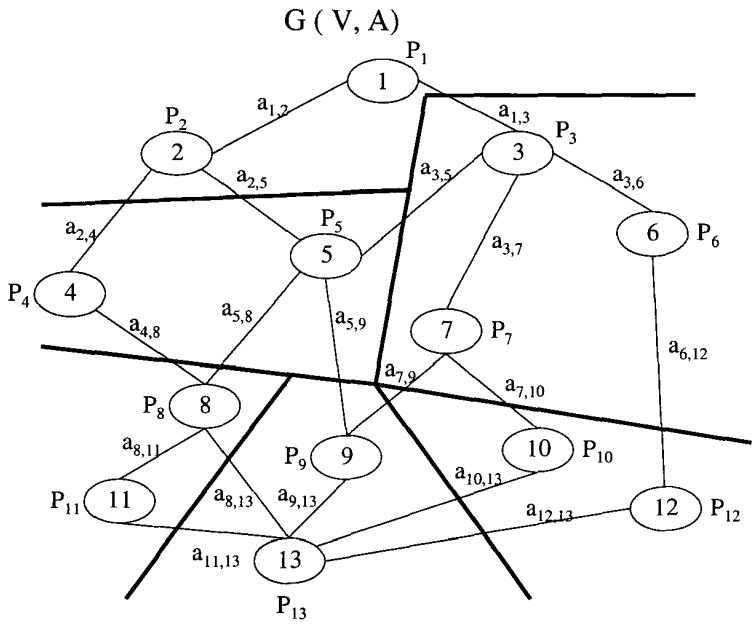


Figura 1.2: Ejemplo del problema de partición de grafos

A es la matriz de adyacencia, en la cual a_{ij} será cero (0) si los vertices i y j no tienen un arco en común, o tendrá el valor del peso del arco que existe entre los vertices i y j.

La función objetivo del problema asocia un valor para cada posible partición del grafo. Así, una posible función objetivo es la siguiente:

$$F_C = \sum_{i,j \in D} a_{ij} + b \frac{\sum_{z=1}^{K} \left(N_{G_z} - \sum_{i=1}^{n} p_i/K\right)^2}{K} \tag{1.1}$$

donde:

$D = \{i \in G_m \ \& \ j \in G_l \ \& \ l \neq m\}$
N_{G_z}: la suma de los pesos de los vertices en el subgrafo G_z
b: factor de equilibrio, $0 \leq b \leq 2$
K: número de subgrafos

El primer término representa el costo de comunicación entre subgrafos, y el segundo el costo por el desequilibrio de carga. El problema consiste en conseguir la partición del grafo que minimize el valor de la función de costo 1.1(función objetivo). Este problema es NP-completo.

1.2.2 Circuito Hamiltoniano Máximo

El problema de encontrar el circuito hamiltoniano de recorrido máximo en un grafo es un problema NP-Completo. Se presentará a continuación la decripción del problema del circuito Hamiltoniano Máximo.

Descripción

Dados N vertices, se debe encontrar el circuito Hamiltoniano de longitud máxima. El circuito hamiltoniano Máximo representa el circuito que involucra los arcos de mayor peso (ver ejemplo en la figura 1.3). Dicho problema se expresa de la siguiente manera:

$$G = (V, A)$$

donde:

$V = \{1, ..., n\}$: conjunto de n vertices,

A es la matriz de adyacencia, en la cual a_{ij} será cero (0) si los vertices i y j no tienen un arco común o tendrá el valor uno (1) si existe un arco entre los vertices i y j.

Además, se define la matriz de pesos de los arcos D:

$$\{d_{ij}\} = \begin{cases} -\infty & \text{Si } a_{ij} = 0 \\ l_{ij} & \text{Si } a_{ij} = 1 \end{cases}$$

donde:

l_{ij}: peso del arco entre los vertices i y j

Suponiendo que los vertices son numerados desde 1 hasta n, una solución al problema puede expresarse a través de una matriz de estado E que indique el orden en que son tomados los vertices para conformar el circuito. Esta matriz tendrá en las filas la posición en el circuito de los vertices y en las columnas los vertices.

$$\{e_{ij}\} = \begin{cases} 1 & \text{Si el vertice j fue el } i\text{-esimo vertice tomado para el circuito} \\ 0 & \text{En otro caso} \end{cases}$$

La matriz de estado E permitirá verificar la validez de una solución, de tal manera de asegurar que todos los vertices son tomados una y solo una vez. Esto se hace con las siguientes restricciones:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} e_{ij} = n \tag{1.2}$$

$$\sum_{i=1}^{n} e_{ij} = 1 \tag{1.3}$$

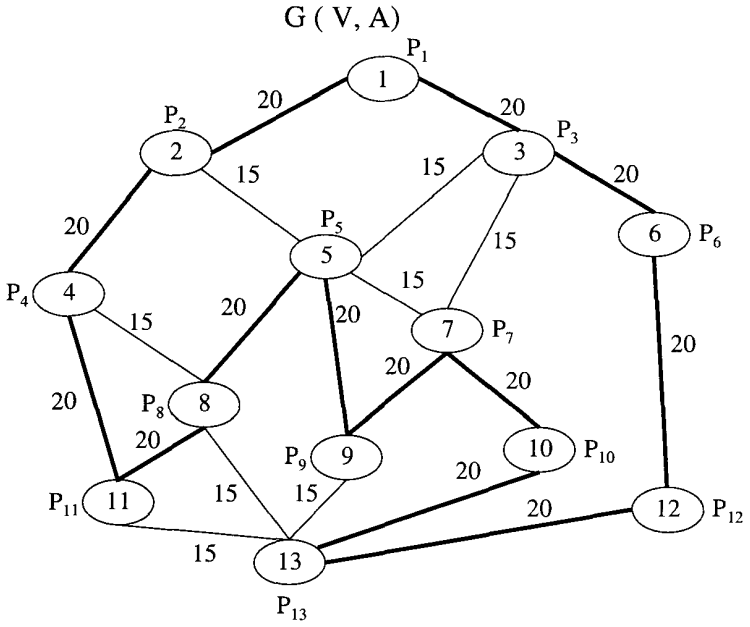


Figura 1.3: Circuito hamiltoniano Máximo

$$\sum_{j=1}^{n} e_{ij} = 1 \tag{1.4}$$

Así mismo, la matriz E permite definir un arreglo unidimensional V de dimensión n, dicho arreglo contendrá en cada posición el vertice que fue tomado para ocupar dicha posición en el circuito.

$$v_j = i \text{ (Si el vertice } j \text{ fue el } i\text{-esimo tomado)}$$

Por último, se propone una función objetivo compuesta por dos partes, una correspondiente a los costos relativos a la suma de los pesos de los arcos (1.5) y la otra relativa al grado de validez de la solución (1.6).

$$F_1 = \sum_{i=1}^{n}\sum_{k=1}^{n}\sum_{j=1}^{n} l_{ij} e_{ij} e_{kj+1} \tag{1.5}$$

$$F_2 = C\left(\left|\sum_{i=1}^{n}\sum_{k=1}^{n} e_{ik} - n\right| + \sum_{i=1}^{n}\left|\sum_{k=1}^{n} e_{ik} - 1\right| + \sum_{k=1}^{n}\left|\sum_{i=1}^{n} e_{ik} - 1\right|\right) \tag{1.6}$$

$$F_C = F1 + F2 \tag{1.7}$$

donde:
C = - Máx($l_{ik} * n$). Representando el factor de penalización por utilizar arcos que no están en la matriz de adyacencia.

El problema consiste en encontrar el recorrido por los vertices que maximice el valor de la función de costo 1.7.

1.2.3 Máximo D-acoplamiento

El problema de acoplamiento puede describirse, en términos de grafos, como el mapeo de los vertices de un grafo en los vertices de otro grafo. Este problema puede ser utilizado para resolver la situación de asignar un conjunto de p procesos (o grupos de procesos) en p procesadores, tomando en cuenta, tanto las comunicaciones de los procesos como los enlaces entre los procesadores. Para eso, se requiere manipular un conjunto de artificios matemáticos que permitan la resolución de los problemas modelándolos a través de D-acoplamiento.

Descripción

El problema de acoplamiento consiste en hallar un conjunto de arcos, los cuales no son adyacente entre ellos, por lo cual, no tienen vertices en común. El problema de máximo $D-$acoplamiento de un grafo G=(V,A) es un subconjunto de A (A^*) tal que para todo v_i que pertenezca a V, el subconjunto de arcos de A^* del cual v_i es una extremidad $A^*(v_i)$, verifica la siguiente restricción: $A^*(v_i) <= D$, de tal forma que ese subgrafo A^* es el de cardinalidad máxima entre todo el conjunto de subgrafos posibles de A. (figura 1.4)

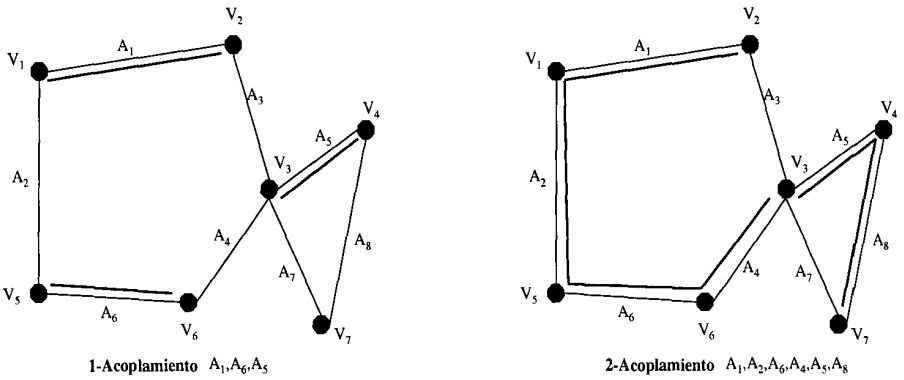


Figura 1.4: Ejemplos de D-acoplamiento

La definicón de problema de D-acoplamiento es la siguiente:
Dado el grafo G

$$G = (\ V,\ E),$$

donde:

$V = \{v_1, ..., V_n\}$ es el conjunto de n vertices,

$E = \{e_1, e_2, ..., e_m\}$ es el conjunto de m arcos del grafo, con cada e_i definido por un par de vertices.

Un D-acoplamiento es un subconjunto de E, llámese E^d, $(e_i, e_j, e_k, ...)$, que cumple la siguiente condición:

Para todo vertice (v_i) del grafo G, se tiene que v_i es extremo, a lo sumo de D arcos del conjunto E^d.

En el problema de asignación de tareas, lo que se desea es hacer un mapeo óptimo de las tareas de una aplicación sobre los procesadores de una plataforma. En estos términos, el problema de máximo D-acoplamiento puede ser utilizado como base para definir un mapeo que maximice el número de enlaces de la plataforma que son utilizados, minimizando el número de arcos del grafo de agrupamiento que deben ser eliminados. A continuación se presenta la descripción detallada del problema según esta descrición:

Dado el grafo G

$$G = (\ V,\ A),$$

donde:

$V = \{1, ..., n\}$ es el conjunto de n procesos (o grupos),

$A = \{a_{ij}\}$ es la matriz de adyacencia,
en la cual a_{ij} será cero (0) si los procesos i y j no se comunican o tendrá el valor del peso del arco que existe entre los procesos i y j.

y el grafo P

$$P = (\ M,\ T),$$

donde:

$M = \{1, ..., n\}$ es el conjunto de n procesadores (vertices),

$T = \{t_{ij}\}$ es la matriz de adyacencia,
que contiene la topología de la plataforma; es decir, t_{ij} será cero (0) si los procesadores i y j no son vecinos o tendrá el valor uno (1) en caso contrario.

El problema de *Dmax*-acoplamiento se convierte en un problema donde se quiere mapear el grafo G al grafo P de manera de usar la mayor cantidad de enlaces de este último. Este problema es un problema combinatorio cuyas soluciones pueden ser representadas por: un **arreglo unidimensional** U, donde el procesador i esta representado por la posición i y el valor de u_i representa el proceso que fue asignado al procesador i, y una función que asigne un valor a cada solución:

$$F_C = \sum_{i=1}^{n} \sum_{k=1}^{n} A_{ij} * T_{u_i u_j} \tag{1.8}$$

De esta manera, el problema consiste en encontrar el máximo para la función 1.8.

Capítulo 2

Técnicas Emergentes

Los problemas involucrados en el desarrollo de un ambiente reconfigurable de procesamiento paralelo/distribuido requieren, en algunos casos, la utilización de técnicas que permitan lograr una buena solución en tiempos razonables. Particularmente, los problemas de optimización combinatoria presentes(partición de grafos, circuito hamiltoniano y D-acoplamiento máximo) requieren del uso de métodos de búsqueda efectivos y eficientes. Como herramienta de resolución de tales problemas se propone el uso de los algoritmos genéticos.

Por otra parte, la complejidad del sistema, y la diversidad de los problemas y de las soluciones particulares, hace necesario que para el diseño del sistema se deba tomar en cuenta la integración y comunicación de módulos aislados e independientes que representan etapas particulares en el alcance de la solución. Estos módulos, integrados, en conjunto consiguen el objetivo final del sistema, el cual es, optimizar el rendimiento de la plataforma y de las aplicaciones. Los Sistemas multiagentes son la herramienta teórica utilizada para resolver tales problemas en el diseño.

Este capítulo contiene una descripción de los algoritmos genéticos, con sus características y mecanismos de funcionamiento. Además, se presenta una introducción de los sistemas multiagentes como herramienta de integración en tales ambientes.

2.1 Algoritmos Geńeticos

Los Algoritmos Genéticos (AG) son algoritmos de búsqueda que emulan la evolución biológica en el computador, siguiendo un proceso *evolutivo inteligente* sobre individuos. Para esto, se basan en la utilización de operadores genéticos tales como mutación, cruzamiento, selección, etc. Estos algoritmos, pertenecen al área de la computación evolutiva, la cual abarca todos aquellos modelos que usan como elemento clave algún mecanismos de la evolución para su diseño e implementación; siendo los algoritmos genéticos los más representativos de dichos modelos.

Los AG fueron propuestos por John Holland a mediados de los anõs 70 [10]. Desde entonces, Holland y sus colaboradores han realizado diversos estudios en la Univer-

sidad de Michigan, con dos objetivos principales: abstraer y explicar rigurosamente los procesos adaptativos de los sistemas naturales, y diseñar sistemas de software que emulen/conserven sus mecanismos .

La idea principal de Holland fue usar características propias del proceso evolutivo natural de las especies sobre un algoritmo programable, para resolver problemas difíciles (NP-Completos). La base fundamental de los AG es simular la evolución de los **individuos** de cierta población, con el objetivo de conseguir mejores individuos.

El procedimiento general de todo AG consiste en mantener una población de potenciales soluciones a través de las generaciones. Cada solución es evaluada usando una función objetivo para determinar que tan buena es. Entonces, se seleccionan las mejores soluciones de la población actual, las cuales serán usadas para ser reproducidas por medio de la aplicación de los operadores evolutivos, lo que permite construir una nueva población.

El objetivo final de los AG es encontrar el mejor óptimo local, partiendo de soluciones escogidas aleatoriamente. Para eso, se utilizan los operadores evolutivos como mecanismos de búsqueda de nuevas, quizás mejores, soluciones. El procedimiento continua hasta cumplirse algún criterio de convergencia o parada, en cuyo caso, el algoritmo debe haber quedado atrapado en un óptimo local [11, 12, 13].

Los campos de aplicación de los AG cubren diversas áreas, principalmente:

- Búsqueda y Optimización.
- Toma de decisiones.
- Clasificación.
- Aprendizaje.

2.1.1 Características de los Algoritmos Genéticos

Los AG poseen características que determinan su funcionamiento y su aplicabilidad a determinados problemas. Estas son las siguientes:

Codificación de Parámetros

Se trabaja con la codificación de un conjunto de parámetros (o un subconjunto de ellos) y no con los parámetros en sí. Dichos parámetros representan, en términos biológicos, los *cromosomas o genes* de los individuos. Los AG requieren que el conjunto de parámetros del problema sean codificados en una cadena de longitud finita sobre algún alfabeto; dicha codificación determina la representación de los individuos. Es posible que para un mismo problema se tengan múltiples formas de codificar los parámetros.

Búsqueda Basada en Población

Se busca desde una población de puntos y no desde uno solo. El concepto que manejan los AG se basa en población, parten de un conjunto de soluciones y no de una solución inicial única, de esta manera, la probabilidad de encontrar un óptimo local malo se reduce.

Función Objetivo como elemento de guía del proceso de optimización

Se usa una función objetivo para definir lo que se desea optimizar, sin requerir más información derivada o auxiliar. Esta función permite evaluar la calidad de los individuos en cada generación. A través de dicha función, es posible medir el grado de adaptabilidad o las aptitudes (*fitness*) del individuo.

Operadores Aleatorios

Se hace uso de operadores estocásticos para las transiciones, en vez de reglas de transición determinísticas. Dichos operadores son simples, involucrando acciones poco complejas, como generación de números aleatorios, copia de cadenas (códigos) e intercambio de segmentos de cadenas.

2.1.2 Macroalgoritmo para los Algoritmos Genéticos

1. Generar una población inicial. Se define el tamaño de la población y se genera aleatoriamente el conjunto de soluciones iniciales para el problema.
2. Evaluar las soluciones. A cada individuo de la población se le aplica la función objetivo para obtener una medida de su calidad.
3. Seleccionar, a través de algún mecanismo, ciertos individuos de la población (soluciones).
4. Reproducir nuevas soluciones mediante la aplicación de los operadores genéticos escogidos sobre el subconjunto de individuos seleccionados.
5. Sustituir algunos individuos de la población según el esquema de reemplazo preestablecido, para introducir en la nueva población individuos generados en la reproducción.
6. Verificar el criterio de convergencia, o regresar al paso 2.

2.1.3 Operadores Genéticos

Los operadores genéticos, llamados también, operadores evolutivos, son los que determinan el cambio de la población durante la ejecución de un AG. Existen básicamente dos operadores o procesos genéticos: mutación y cruzamiento. Sin embargo, han sido

propuestos otros operadores que se acoplan a problemas particulares, bien sea por la definición de nuevos operadores o por modificaciones a los ya existentes. A continuación se describen los operadores genéticos básicos y algunos operadores especiales.

Mutación

El operador de mutación es un operador unario que simula el proceso evolutivo que ocurre en los individuos cuando cambia su estructura genética. Cuando sobre algún individuo se aplica mutación, se seleccionan aleatoriamente componentes (genes) de dicho individuo para ser modificados, también de forma aleatoria. Esto hace que a través del operador de mutación se puedan producir cambios en la estructura de un individuo que hagan que éste tenga cierto grado de adaptabilidad, al permitir que, eventualmente, aparezca nueva información.

Cruzamiento

El cruzamiento o mezcla es un operador, normalmente binario, que permite expresar el proceso de apareamiento natural usando operaciones sencillas. Mediante el operador de cruzamiento se toman diversos componentes de distintos individuos para generar con ellos un nuevo individuo, el cual heredará características de sus padres. Su grado de adaptabilidad dependerá de la combinación de esas características.

Operadores Avanzados

En algunas circunstancias particulares, los operadores genéticos básicos no son suficientes para hallar una solución satisfactoria a un problema dado, bien sea porque conducen a una convergencia prematura del algoritmo, o porque no pueden ser utilizados en la codificación de los parámetros seleccionados, o porque se desean explorar otros mecanismos de diversificación de la población. Para estos casos, se han planteado operadores avanzados cuya aplicación depende de los problemas particulares. Algunos de estos operadores son:

- **Dominancia**. Determina algunos componentes (genes) que son preponderantes en el individuo para su valor de *fitness*, y da preferencia a esos componentes al aplicar otros operadores genéticos, de tal manera de tender a que sean preservados de generación en generación.

- **Segregación**. Es lo contrario a la dominancia. Los componentes segregados no influyen de manera determinante en el grado de adaptabilidad del individuo.

- **Inversión**. Este operador modifica la información genética de un individuo tomando dos componentes (genes) de él mismo e intercambiando los valores de estos; es decir, al primero le asigna la información del segundo y al segundo la información del primero.

- **Duplicación**. Copia exactamente la información de un individuo en otro.

2.2 Sistemas Multiagentes

El paradigma de Agentes emerge del área de Inteligencia Artificial. Los Sistemas Multiagentes cubren una diversidad de aspectos considerados desde diferentes puntos de vistas (reactividad, cognición, etc.), por lo que han sido usados como técnica integradora de otras áreas (Sistemas Expertos, Redes Neuronales Artificiales, etc.) para la resolución de diferentes problemas. Los sistemas multiagentes no son mas que sistemas donde sus actividades son el fruto de la interacción entre entidades relativamente autónomas e independientes, llamadas agentes, que trabajan en su seno según modos complejos de cooperación, de conflicto y de concurrencia, lo que les permite sobrevivir y perpetuarse. Estos sistemas enriquecen el área de Inteligencia Artificial al usar metáforas sociológicas, tales como de cooperación, de negociación, de grupo y de equipo; y de la biología, tales como auto-organización, adaptación, etc. [5].

Entre las ventajas del enfoque multiagentes pueden señalarse:

- Los problemas son resueltos con mayor rapidez, permiten el aprovechamiento del procesamiento paralelo.
- Tienen mayor flexibilidad, pues se tienen agentes con diferentes habilidades que en forma dinámica cooperan entre si para resolver el problema.
- Es más confiable, pueden tenerse agentes que tomen las responsabilidades de otros en caso de fallas.

2.2.1 Caracterización de los Agentes

Es difícil encontrar en la literatura una definición de Agente que sea el consenso. Aquí, se presentará la de Ferber [14]: *Un agente es un hardware o software que puede actuar sobre si mismo o sobre su ambiente. Además, tiene una representación parcial de su ambiente y puede comunicarse con otros agentes. Por otro lado, tiene objetivos individuales y su comportamiento es el resultado de las observaciones, conocimiento, habilidades e interrelaciones que el puede tener con otros agentes o con su ambiente.* En general, un agente puede ser descrito por:

- *Sus Tareas:* es decir, el conjunto de actividades de los agentes, las cuales le permiten cumplir sus objetivos y/o funciones.
- *Sus Conocimientos:* es decir, las reglas que siguen los agentes para realizar sus tareas. Dicho conocimiento puede ser adquirido según alguna de las siguientes técnicas: especificadas por el diseñador (a través de algún formalismo), incorporadas dinámicamente durante su evolución, proveniente de otras fuentes de conocimiento (agentes, etc.), o derivarse de sus propios mecanismos de aprendizaje.

- *Su Comunicación:* definen su forma de interacción con el ambiente y con los otros agentes, por consiguiente, se deben definir los lenguajes, protocolos de comunicación, etc.

La Figura 2.1 muestra un modelo de agente que abarca los aspectos mencionados anteriormente.

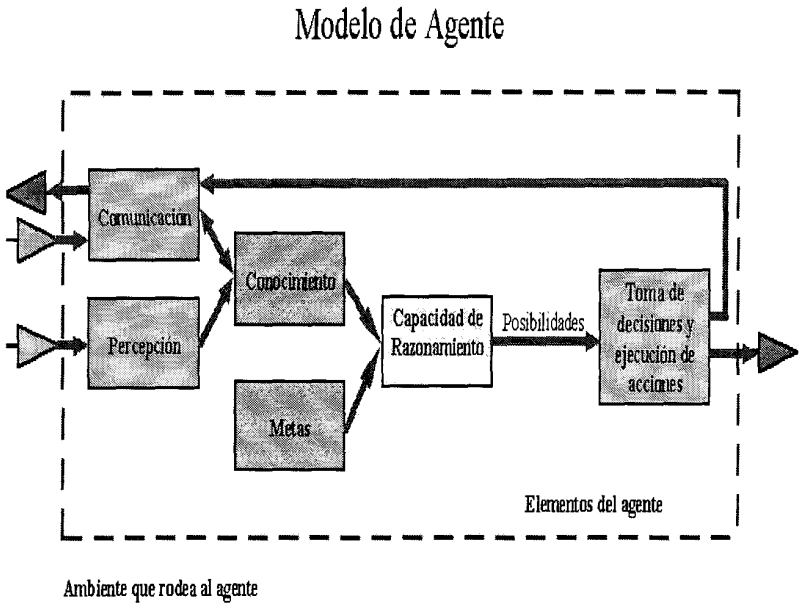


Figura 2.1: Modelo de un Agente

Los agentes realizan un ciclo continuo de observación y acción. Perciben la información a través de sus sentidos y la transforman en conceptos que les permiten definir la situación en la cual se encuentran. De esta manera, comprueban si el objetivo perseguido ha sido alcanzado, y en caso de una respuesta negativa deciden, según ciertos criterios, cuales acciones deben ejecutarse para lograr la meta, y a continuación, ejecutan dichas acciones.

Desde el punto de vista de implementación u operación, los agentes son objetos que describen los desarrolladores. Se asume que los agentes representan entidades del modelo, tales como procesadores, aplicaciones, controladores, etc. De esta manera, cuando se escribe un código para un agente, se deben considerar las siguientes partes:

- La estructura de datos, la cual tendrá las variables de estado del agente.
- Las funciones (acciones) que manipularan la información que recibe el agente.

Además, se especifican las propiedades o características que posee el agente, las cuales pertenecen al siguiente conjunto (figura 2.2):

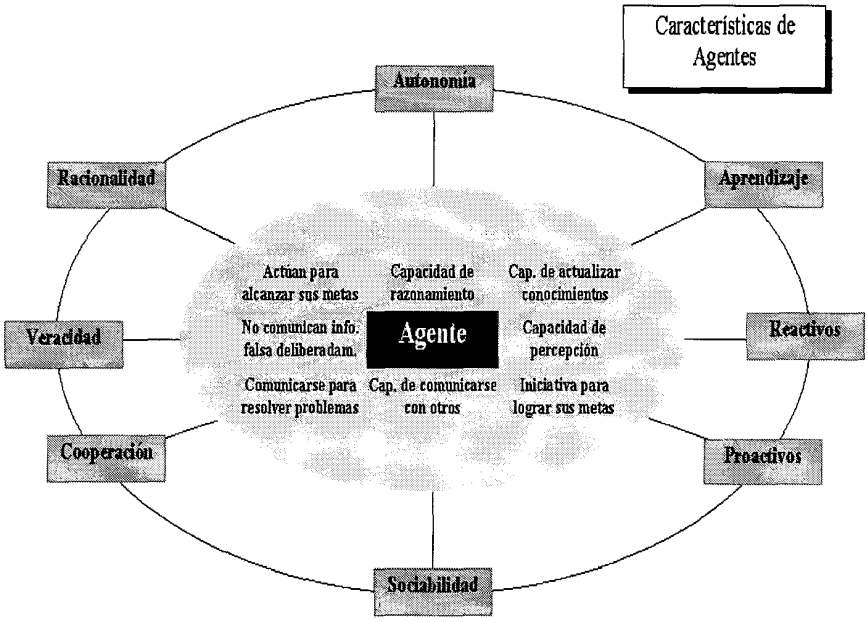


Figura 2.2: Características de un Agente

- *Autonomía:* operación sin la intervención directa de algún ente externo.
- *Cooperación y Colaboración (sociabilidad):* capacidad para cooperar y colaborar con otros agentes, lo que conlleva a acciones de comunicación e interacción. La cooperación esta basada en la coordinación de actividades elementales. Mientras, la colaboración se basa mas en la noción cliente/servidor, pudiéndose establecer contratos y requerimientos (propuestas) de servicios.
- *Movilidad:* Moverse (su código y datos) a un sitio remoto, es decir, trasladarse desde un sitio a otro acumulando conocimiento e información.
- *Inteligencia y aprendizaje:* un agente puede realizar actividades cognitivas. Dichas actividades pueden consistir desde una simple regla de inferencia hasta el cumplimiento de objetivos con mecanismos de auto-aprendizaje.
- *Asincronismo:* El comportamiento de un agente puede regirse por acciones gobernadas, solamente, por sus propias reglas.

- *Comportamiento Proactivo:* puede tener sus propios objetivos y actuar en base a ellos, no solamente como respuestas a estimulos externos.
- *Comportamiento reactivo:* Los agentes perciben su ambiente (mundo real, otros agentes, usuario, etc.) y responden a los cambios de éste.
- *Veracidad:* Se supone que un agente no debe comunicar, en forma deliberada, informacin falsa.
- *Racionalidad:* El agente actuará para alcanzar sus metas en la medida en que sus creencias, su conocimiento y su capacidad de razonamiento se lo permita.

2.2.2 Metodología de Diseño

El diseño de sistemas multiagentes comparte algunos aspectos relativos al diseño genérico de sistemas distribuidos. El objetivo principal es plantear un sistema que responda a ciertos requerimientos dados en circunstancias diversas. La metodología que se propone contiene los siguientes pasos:

Descripción del sistema

En esta etapa se especifican las actividades del sistema y los problemas que deben ser resueltos; describiendo los componentes del sistema que pueden ser representados como agentes con su respectiva presentación genérica. Además, se propone la arquitectura del sistema.

Análisis Funcional

En este etapa se describen, de manera detallada, las funciones de cada uno de los agentes del sistema; presentándose como actividades propias o como servicios a otros agentes del sistema.

Análisis Comportamental

Para cada agente, se presentan sus capacidades, niveles de conocimiento y sus interrelaciones con otros agentes. Con esta información se tendrá una visión de los agentes pudiéndose clasificar de acuerdo a su comportamiento.

Análisis Comunicacional

En esta etapa se detallan los enlaces e interrelaciones entre los agentes a través de sus necesidades de comunicación. Se pueden establecer relaciones tipo cliente-servidor, cooperación, conflicto, etc

Capítulo 3

Propuesta del Sistema Manejador del Ambiente

A continuación se presenta el esquema global de funcionamiento del SMA, y de los módulos que lo componen. Además, se describe detalladamente el núcleo administrador del sistema.

3.1 Funcionamiento del SMA

El funcionamiento del sistema esta compuesto por las tres etapas siguientes: caracterización del sistema y de las aplicaciones, reconfiguración del sistema y asignación de las tareas. (figura 3.1)

Figura 3.1: Módulos del SMA

3.1.1 Caracterización del sistema y de las aplicaciones

Una aplicación paralela/distribuida puede modelarse mediante un grafo $G_p = (V_p, E_p)$ donde los vertices representan las tareas (procesos) y los pesos de los vertices representan los tiempos de ejecución (estimados o conocidos) para esas tareas(ver figura 3.2). Los arcos representan los requerimientos comunicacionales entre los procesos, y los pesos de dichos arcos los comunicacionales o la cantidad de información a transferir entre ellos. Este esquema de representación asume que el grafo de la aplicación es estático, puesto que, no hay generación dinámica de procesos.

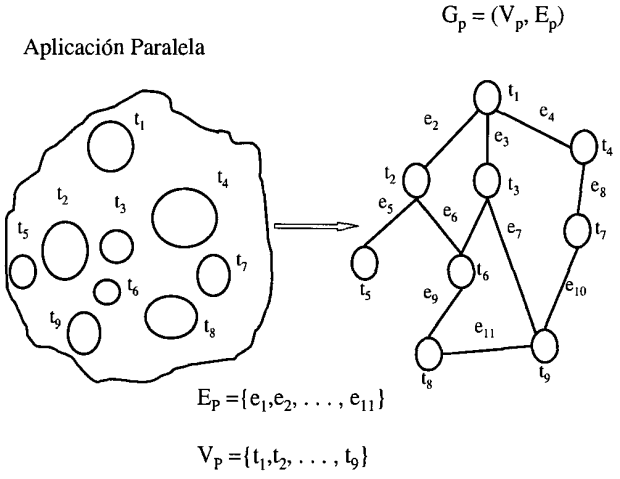


Figura 3.2: Representación de la aplicación

Bajo un esquema similar, la arquitectura paralela se modela a través de un *grafo conexo no dirigido* $G_t = (V_t, E_t)$, donde los vertices representan procesadores y los arcos enlaces de comunicación entre los mismos, esto se ilustra con el ejemplo de la figura 3.3.

De esta manera, el sistema quedará descrito por:

- Las características de la arquitectura paralela: número de enlaces máximo por procesador (D_{max}), número de procesadores en el sistema (K), topología de interconexión de los procesadores (G_t), etc.
- El número de tareas de los programas (n) y el flujo de ejecución de las tareas, conocido como el grafo de los procesos(G_p)

3.1.2 Reconfiguración del Sistema

La reconfiguración del sistema es la parte fundamental en la propuesta, la cual comprende la agrupación de los procesos y el acoplamiento (mapeo) del grafo de los procesos

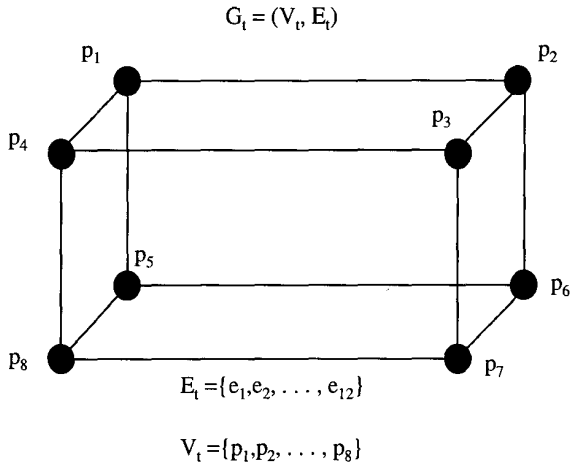


Figura 3.3: Ejemplo de representación de la arquitectura mediante grafos

sobre la topología de interconexión de la plataforma. La reconfiguración se está entendiendo como un proceso que de forma transparente realiza las transformaciones necesarias para optimizar el rendimiento de la aplicación sobre la plataforma. Tal y como se ha enfocado el trabajo, no se realizan transformaciones sobre la plataforma (reconfiguración por *hardware*) sino sobre la aplicación. Se ha separado el funcionamiento de este módulo en dos etapas como se muestra en la figura 3.4; una de análisis estático y otra de acoplamiento topológico.

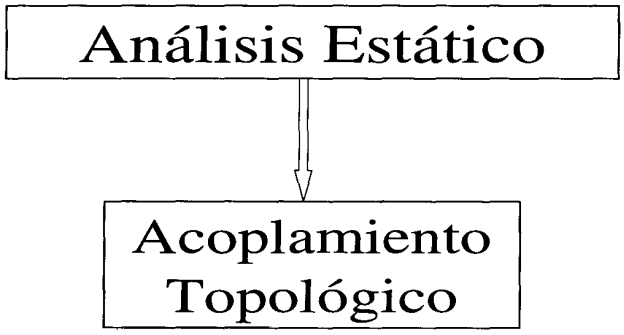


Figura 3.4: Etapas del módulo de reconfiguración

3.1.3 Asignación de las tareas

Se asignarán las tareas a los procesadores según lo indicado por el grafo de agrupamiento, creándose y ejecutándose los procesos de control necesarios, lo cual involucra la creación de una tabla de procesos para la aplicación y una entrada en la tabla general de aplicaciones para el sistema. Es en esta etapa donde cada proceso es enviado a un procesador, registrándose la información referente a la ubicación, tiempo de creación, estado, etc.

3.2 Núcleo administrador del sistema

Se ha considerado como núcleo administrador del sistema la etapa encargada de la reconfiguración del mismo, la cual debe resolver los problemas de agrupación de las tareas (partición de grafos) y acoplamiento de la topología del grafo de ejecución a la plataforma.

El SMA tiene dos alternativas para el desarrollo del núcleo administrador. Una primera alternativa trata de adaptar el grafo de ejecución de las tareas a la topología real del sistema, y una segunda alternativa genera una topología virtual de interconexión de los procesadores la cual se adapta al grafo de ejecución de las tareas. En este último caso, se debe hacer la gestión de esas topología virtual.

3.2.1 Esquema general

El esquema general de funcionamiento del núcleo administrador consta de las dos etapas en las que consiste el módulo de reconfiguración del sistema. Estas son:

- *Etapa de Análisis Estático:.* En esta etapa se determinan los parámetros de evaluación de la función objetivo (Costos de Comunicación, de Ejecución, etc).
- *Etapa de Acoplamiento Topológico:* En esta etapa, según la alternativa escogida, se pasa a la fase de adaptar al sistema o al programa:
  - Primera Alternativa: (adaptación de la aplicación)
    - *Agrupación de tareas:* tal que el número de grupos de tareas sea el mismo que el número (K) de procesadores en el sistema. Durante este proceso, se debe optimizar la función objetivo [15]. Este nuevo grafo se denominará grafo de agrupamiento G_p^1.
    - *Verificación de las Restricciones del Sistema Físico:* en este caso, es necesario eliminar el menor número de arcos del grafo de agrupamiento, tal que se adapte al número máximo de enlaces (D_{max}) que puede tener cada procesador del sistema. Además, se debe asegurar que el grafo de agrupamiento sea conexo para garantizar que no existen grupos de tareas aislados.

- Segunda Alternativa: (adaptación del sistema)
  * *Agrupación de tareas:* Esta fase es la misma que para la otra alternativa.
  * *Generación de la Topología Virtual:* Una vez que se tiene el grafo de agrupamiento, se genéra la topología virtual sobre la plataforma computacional que satisface los requerimientos del grafo.

3.2.2 Especificación detallada de las alternativas

Como se mostró anteriormente, el núcleo administrador esta dividido en dos partes, sin importar cual sea la alternativa. La primera parte, que consiste en *agrupar las tareas*, se resuelve usando los algoritmos propuestos en [16, 17], en los cuales se describen varias heurísticas paralelas basadas en Algoritmos Genéticos para resolver problemas de Optimización Combinatoria. Cada una de las heurísticas presentadas en esos trabajos explotan diferentes conceptos del paradigma de programación paralela, tales como descomposición del espacio de datos, explotación del paralelismo implícito, incorporación de conceptos ligados a la inteligencia colectiva, etc. Así, la agrupación de tareas es modelada como un problema de partición de grafo, donde los vertices del grafo representan las tareas, los arcos representan las comunicaciones entre las tareas y k (número de particiones) representa la cantidad de procesadores.

La segunda parte, en la cual se intenta mapear el grafo de ejecución sobre la topología puede ser desarrollada bajo dos enfoques distintos, el primero basado en la verificación de las restricciones del sistema, y el segundo basado en la topología virtual. En ambos casos, el mapeo del grafo de agrupamiento sobre la topología de interconexión se convierte en un algoritmo de asignación. En el caso de verificación de las restricciones del sistema se presentan dos alternativas para dar mayor flexibilidad y adaptabilidad al sistema.

Esquema basado en la verificación de las restricciones del sistema

La parte de *verificación de las restricciones del sistema* es compleja. En este caso, lo que se busca es que el grafo de agrupamiento de tareas sea lo más parecido al grafo de interconexión de los procesadores del sistema, maximizándose el número de arcos del grafo de agrupamiento. Este problema puede ser definido como un problema de *máximo* D_{max}*−acoplamiento*, en el cual, lo que se busca es minimizar el número de arcos eliminados del grafo de agrupamiento tal que el grado de cualquier nodo de ese grafo sea menor o igual a D_{max}. Además, el grafo tiene que ser conexo. Este problema, con esta última restricción es NP-complejo. Se proponen dos algoritmos para resolverlo:

1. **Primer Algoritmo:**

   - Buscar el *Circuito Hamiltoniano máximo* del grafo de agrupamiento (en este caso, se supone que el grafo de la plataforma tiene un circuito hamiltoniano). El ciclo encontrado será el nuevo grafo de agrupamiento G_1^a.

- Agregar arcos del grafo de agrupamiento a G_1^a mientras que el grado de cualquier nodo de este grafo sea menor o igual a D_{max}, tratando de minimizar a la vez, el costo de comunicación en el sistema.
- Asignar los vertices de G_1^a a los vertices de P.

2. **Segundo Algoritmo:**

- Buscar un $D_{max}-$*acoplamiento de cardinalidad máxima* para el grafo de agrupamiento, es decir, con el mayor número de arcos posibles. Esto da el borde superior de la configuración final del grafo de agrupamiento.
- Añadir arcos tal que se asegure que el grafo de agrupamiento sea conexo y que no se viole la restricción de que cualquier v_i del grafo final tenga un grado menor o igual a D_{max}.
- Asignar los vertices de G_1^a a los vertices de P.

Esquema basado en topología virtual

El algoritmo basado en topología virtual trata de asignar los nodos (vertices) del grafo de agrupamiento sin necesidad de realizar eliminaciones de arcos. Así, todos los arcos del grafo de agrupamiento permanecerán en la asignación definitiva. Esto se logra estableciendo enlaces entre nodos no vecinos del grafo de la plataforma. El macroalgoritmo que describe el funcionamiento de este esquema se presenta a continuación:

1. Seleccionar el vertice de mayor grado v_{max} (aleatorio si hay más de uno) del grafo de agrupamiento G^a.

2. Asignar v_{max} al vertices de mayor grado del grafo de la plataforma p_{max}.

3. Asignar los vecinos de v_{max} ($S(v_{max})$) a los vertices vecinos de p_{max} ($S(p_{max})$). Si la cardinalidad de $S(v_{max})$ es mayor que la cardinalidad de $S(p_{max})$, la diferencia se asignará a la vecindad de $S(p_{max})$.

4. Eliminar v_{max} de G^a (los arcos que inciden en v_{max} también serán eliminados).

5. Seleccionar un nuevo v_{max} de $S(v_{max})$, y el nodo donde este es asignado como el nuevo p_{max}.

6. Repetir el proceso desde 3 hasta que todos los vertices de G^a hayan sido asignados a un vertice de P.

3.2.3 Resolución de los Problemas NP-Completos asociados a las alternativas

El núcleo administrador del sistema propone un esquema de funcionamiento de dos etapas. En la primera etapa es necesario generar el grafo de agrupamiento de las tareas, el cual se describe a través del problema de partición de grafos. Para la segunda etapa se presentan dos esquemas, uno basado en la verificación de las restricciones del sistema y el otro en topología virtual. Bajo el primer esquema, se debe resolver alguno de los problemas siguientes: **Circuito Hamiltoniano Máximo** o **Máximo D-acoplamiento** . A continuación se describe el mecanismo de resolución empleado en cada caso:

Agrupamiento de las tareas

El agrupamiento de las tareas es modelado a través de la teoría de grafos como un problema de particionamiento de un grafo. Así, la solución al problema de agrupamiento de las tareas es resuelto usando la descripción presentada en la sección 1.2.1. Lo que se busca es encontrar un agrupamiento que minimice tanto la comunicación entre grupos como el desbalance de la carga. Para la implementación de la solución se usa el esquema propuesto en [16], el cual esta basado en algoritmos genéticos.

Circuito Hamiltoniano Máximo

La descripción de este problema se presentó en la sección 1.2.2. En términos del sistema, lo que se quiere es encontrar un circuito Hamiltoniano de longitud máxima en el grafo de agrupamiento con el objeto de mapear directamente la mayor cantidad posible de comunicaciones sobre la topología de la plataforma. Esta alternativa puede aplicarse siempre y cuando la plataforma tenga un circuito Hamiltoniano. Tomando como base el algoritmo genético propuesto en [16], se describe la solución a este problema:

1. Se toma como grafo de entrada el grafo de agrupamiento generado en la etapa 1.

2. Se aplica un algoritmo genético con la función objetivo descrita por la ecuación 1.5

3. La mejor solución de este algoritmo se toma como el circuito Hamiltoniano máximo.

4. El circuito Hamiltoniano máximo se mapea sobre el circuito Hamiltoniano de la plataforma tomando como puntos de inicio el procesador con mayor número de enlaces y el nodo del grafo de agrupamiento con mayor volumen de comunicación. Así, se realiza la asignación de cada nodo del grafo de agrupamiento a un procesador de la plataforma.

Máximo *D*-acoplamiento

El acoplamiento máximo tiene como objetivo el uso máximo de los enlaces de la plataforma. Tal como se describe en la sección 1.2.3 este es un problema NP-Completo, cuya solución requiere la aplicación de un algoritmo que genere una buena solución en un tiempo razonable; estos requerimientos son cubiertos por los algoritmos genéticos de una manera directa y sencilla. Para este caso, los individuos del AG representarán soluciones al problema de máximo D-acoplamiento. Para la codificación de los individuos se usará un arreglo de valores enteros, donde la posición i del arreglo representa al procesador i y el valor del arreglo en la posición i contiene el nodo del grafo de agrupamiento que ha sido asignado a al procesador i.

A continuación se presenta el macroalgoritmo basado en AG propuesto para resolver este problema:

1. Se toma como grafo de entrada el grafo de agrupamiento generado en la etapa 1.
2. Se genera una población de soluciones iniciales, cada una de las cuales representa una solución al problema de máximo D-acoplamiento.
3. Se aplica un AG función objetivo 1.8.
4. Se toma la mejor solución del AG como la asignación de los nodos del grafo de ejecución a los procesadores de la plataforma.

Capítulo 4

Modelado del Sistema Manejador del Ambiente basado en Sistemas Multiagentes

El capítulo 3 describió el funcionamiento global del Sistema Manejador del Ambiente y del núcleo fundamental del mismo. Sin embargo, en el mismo no se especifican los aspectos de integración de los módulos y de las etapas que componen el SMA. Estos elementos del diseño serán abordados a través de la teoría de Sistemas Multiagentes. El presente capítulo describe el diseño detallado del SMA basado en Sistemas Multiagentes.

4.1 Descripción del Sistema Multiagentes

El estudio que se propone en esta parte abarca aspectos que tienen que ver con los problemas de asignación de tareas, balance de carga, acoplamiento topológico y optimización de requerimientos comunicacionales en aplicaciones que están compuestas por tareas que cooperan (paralelamente o concurrentemente). La visión multiagentes del problema permite plantear un esquema donde se define una nueva estructura del SMA cuyos componentes estén representados por agentes con diversas capacidades y características (ver figura 4.1). El sistema esta compuesto por los siguientes agentes:

- Agentes decisorios: Poseen conocimiento de la configuración de la plataforma y de su estado global. Tiene una interfaz con la aplicación que le permite tomar decisiones relativas al tipo de planificación que se usará para dicha aplicación. Cumplen, entre otras actividades, el papel del módulo de caracterización del sistema y de las aplicaciones.

- Agentes planificadores: Realizan las actividades asociadas con el núcleo administrador del sistema (módulo de reconfiguración). Existen dos tipos de Agentes Planificadores, inspirados en lo definido en la sección 3.2.1, según quien se adapte a quien (aplicación a plataforma, o viceversa).

- Agente ejecutores: agentes que corren las tareas que componen la aplicación. Existirán tantos agentes ejecutores como unidades de procesamiento tenga la plataforma, pudiendo cada uno de ellos ejecutar más de una tarea. Los agentes ejecutores cumplen parte de las actividades enmarcadas en la etapa de asignación de las tareas, puesto que son los entes ejecutores de esas tareas.

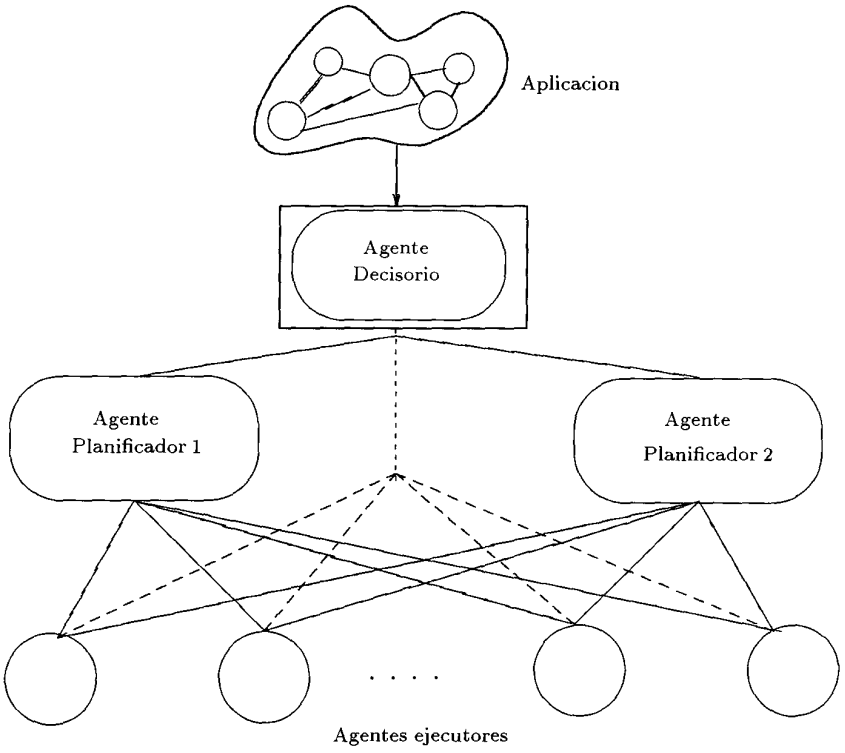


Figura 4.1: Arquitectura del SMA basado en Sistemas Multiagentes

En esta propuesta, la aplicación interactua directamente con algún agente decisorio, el cual según parámetros del sistema y de la aplicación, solicitará a uno de los agentes de planificación la asignación de las tareas sobre los agentes ejecutores. Como se dijo antes, la existencia de dos tipos de agentes planificadores se debe a la utilización de la propuestas presentadas en el capítulo 3, donde se definen dos esquemas de acoplamiento topológico. Finalmente, los agentes ejecutores ejecutan los programas que se les asignan.

4.2 Análisis funcional

A continuación se presenta un descripción detallada de las funciones que cumplirá cada tipo de agente en el sistema.

Agente decisorio

- Capturar y Analizar el estado global del sistema.
- Analizar los requerimientos comunicacionales de la aplicación (paradigma de programación paralela).
- Tomar la decisión del tipo de planificación a usar.
- Solicitar la ejecución por el planificador "adecuado"

Agente planificador tipo 1

- Agrupar las tareas para que el número de tareas sea menor o igual al número de agentes ejecutores.
- Verificar las restricciones del sistema, para lo cual se deben eliminar arcos del grafo de agrupamiento tal que se adapte al número máximo de enlaces que tiene cada agente ejecutor.

Agente planificador tipo 2

- Agrupar las tareas para que el número de tareas sea menor o igual al número de agentes ejecutores.
- Establecer enlaces entre los agentes ejecutores para configurar una topología virtual de interconexión que se acople al grafo de agrupamiento de la aplicación.

Agentes ejecutores

- Ejecutar las tareas que le sean asignadas.
- Registrar y reportar su estado al agente decisorio.

4.3 Análisis comportamental

El funcionamiento del sistema multiagentes esta determinado, fundamentalmente, por el comportamiento de los agentes que lo componen. Dicho comportamiento se basa en las capacidades que tengan los agentes, los estímulos que reciban, y el conocimiento que estos tengan, tanto del medio como de ellos. Estos aspectos definen, de manera general, la estructura del agente.

Agente decisorio

Recibe cada solicitud de ejecución de una aplicación, la cual viene modelada por su flujo de ejecución. Además, decide a que planificador enviará la solicitud y espera por respuesta. La estructura de un agente decisorio se describe a continuación:

- **Capacidades**
  Realiza el modelado y memorización del APPD. Además, verifica las restricciones y toma la decisión sobre el tipo de planificación a usar.
- **Conocimiento**
  Almacena el estado global del sistema y clasifica las aplicaciones según los paradigmas de programación (árbol, divide y ejecuta, encauzamiento, etc.). Adicionalmente, almacena información del pasado de los resultados obtenidos para ciertas aplicaciones bajo una determinada planificación. Esta información es la que utiliza en el proceso de decisión.
- **Conectividad**
  Por ser el agente que modela el sistema, requiere comunicación con todos y cada uno de los agentes que componen el SMA.

Agente planificador

El agente planificador actúa de manera procedimental, recibe la solicitud de planificación del agente decisorio y aplica a esta un procedimiento (diferente para cada tipo de planificador), del cual obtiene el esquema de asignación de tareas. Luego, ordena a los ejecutores la corrida de las tareas. La estructura del agente planificador es la siguiente:

- **Capacidades**
  Su principal capacidad es de organización. Este agente planifica, coordina, asigna y controla la ejecución de las tareas de la aplicación.
- **Conocimiento**
  Los agentes planificadores mantienen una tabla de control de las tareas en ejecución, y otra con la asignación propuesta.
- **Conectividad**
  Se comunican con el agente decisorio y con los ejecutores.

Agentes ejecutores

Los ejecutores reciben la orden de correr una determinada tarea. Además, reportan el estado de la tarea y el suyo. Los agentes ejecutores presentan la siguiente estructura:

- **Capacidades**
  Los ejecutores son agentes que realizan las operaciones (corren las tareas).
- **Conocimiento**
  Conocen su capacidad de computo y almacenan estados (propios y de las tareas que ejecutan). Adicionalmente, mantienen una estructura de conexión que indica si pueden o no establecer comunicación con otros ejecutores.

- **Conectividad**
  Se comunican con los agentes decisorios y planificadores indicándoles su estado. Además, reciben ordenes de los agentes planificadores. Por otro lado, se interconectan con el resto de los agentes ejecutores para permitir la comunicación entre las tareas.

4.4 Requerimientos comunicacionales

Los requerimientos comunicacionales del SMA se expresan en términos del intercambio de información entre los agentes, tanto para la ejecución de la aplicación, como para la actualización del estado del APPD. Las comunicaciones necesarias en el SMA se presentan a continuación:

- **Aplicación - Agente decisorio**. Para este caso, se asume la aplicación como un agente con una estructura particular que solicita el servicio de ejecución al agente decisorio, y recibe de este alguna respuesta.

- **Agente decisorio - Agente planificador**. Una vez que el agente decisorio verifica las condiciones del sistema y los requerimientos de la aplicación, envía la solicitud al agente planificador y recibe un reporte del plan de ejecución generado. Además, el agente decisorio recibe del planificador los resultados de la ejecución de la aplicación.

- **Agente decisorio - Agente ejecutor**. La comunicación entre los agentes ejecutores y el agente decisorio tiene como objetivo el control del estado global del SMA. Los ejecutores reportan, periódicamente, su estado al agente decisorio. También, este último puede solicitar un reporte asíncrono.

- **Agente planificador - Agente ejecutor**. La transferencia de información entre el(los) planificador(es) y los ejecutores esta constituida por el ordenamiento de ejecución de tareas por parte del agente planificador al agente ejecutor, según el plan de ejecución generado por el primero. Adicionalmente, el agente ejecutor reporta el estado de las tareas (ej. finalización de tarea).

- **Agente ejecutor - Agente ejecutor**. El mecanismo de comunicación entre los agentes ejecutores será dinámico; el cual estará determinado por la topología del APPD y/o los enlaces virtuales que se establezcan para satisfacer los requerimientos de las topologías virtuales. De esta manera, se podrán realizar comunicaciones entre las tareas.

Capítulo 5

Pruebas y Resultados

En este capítulo se presentan las pruebas y el análisis de los resultados. Para tal fin, se describen las aplicaciones especificando sus estructuras, y los parámetros usados en las pruebas. Las pruebas se dividieron en dos partes. La primera verifica los esquemas propuestos en la sección 3.2. Mientras, la segunda parte esta basada en un simulador de eventos discretos que permite evaluar el comportamiento del SMA (tiempos de respuesta, rendimiento, etc.) modelado en términos de la teoría de agentes.

5.1 Descripción de las aplicaciones

Desde el punto de vista experimental, es necesario caracterizar las aplicaciones a las cuales el sistema dará servicios. Así, deben definirse los mecanismos de representación, los tipos de aplicaciones y algunos valores que reflejen el volumen esperado de aplicaciones en el sistema.

5.1.1 Estructura de las aplicaciones

Las aplicaciones se representaran a través de un grafo como se especificó en la sección 3.1.1. Para la implementación, se tiene un arreglo de n elementos, cada uno de esos elementos es una tarea con el tamaño de la misma (p.e: su tiempo de ejecución). La matriz de adyacencia se puede convertir en una matriz de pesos, lo cual indicará el tiempo que se requiere para la comunicación entre dos tareas dadas. Un valor de cero (0) en dicha matriz indicara que no existe comunicación entre esas dos tareas.

5.1.2 Tipos de aplicaciones

La aplicaciones se clasifican tomando en cuenta tres parámetros, cuyos valores pueden variar según tres rangos: alto, medio y bajo. Los parámetros y sus rangos son:

- Número de tareas (**NT**). Cuantos procesos, concurrentes o no, componen la aplicación.

alto: (50 - 70), medio: (20 - 30), bajo: (5 - 10)

- Tamaño promedio de las tareas (**TPT**). Tiempo promedio de ejecución, en minutos, de las tareas en un procesador de velocidad relativa 1.
  alto: (40 - 60), medio: (15 - 25), bajo: (1 - 8)

- Volumen de comunicación promedio por tarea (**VCP**). Porcentaje de tiempo, respecto al tiempo de ejecución, que se emplea para comunicar dos tareas dadas.
  alto: (30 - 50), medio: (15 - 20), bajo: (2 - 10)

Tomando en cuenta los parámetros anteriores y las aplicaciones comunes en un APPD, se han definido los siguientes tipos de aplicaciones:

1. Grano fino con muchos procesos y poca comunicación. (NT= alto, TPT=bajo, VCP=bajo)

2. Grano fino con alta dependencia de datos. (NT= alto, TPT=bajo, VCP=alto)

3. Grano medio con baja comunicación. (NT= media, TPT=media, VCP=bajo)

4. Grano medio con alta dependencia de datos. (NT= media, TPT=media, VCP=alto)

5. Grano grueso con muchas tareas y poca dependencia de datos. (NT= alto, TPT=alto, VCP=bajo)

6. Grano grueso con pocas tareas y muy poca comunicación. (NT= bajo, TPT=alto, VCP=bajo)

Además, se define un parámetro que refleja la cantidad de enlaces que tiene cada tarea; el cual permite medir el volumen de arcos del grafo de la aplicación y la uniformidad de grafo (en cuanto a número de arcos por nodo). De esta manera, se describen tres grupos:

Homogéneos con un número bajo de arcos por tarea (2-4)

Homogéneos con un número alto de arcos por tarea (6-8)

Heterogéneos con un número de arcos por tarea que va entre mediano y alto (5 - 10)

Con este parámetro adicional, se generan tres categorías por cada tipo de aplicación.

5.1.3 Tasas promedio de llegada de aplicaciones

El tiempo entre llegada de las aplicaciones es un parámetro fundamental para medir el estado del sistema; puesto que, determina en una buena medida la carga del mismo. Por lo tanto, se proponen valores distintos para las tasas de llegadas de las aplicaciones que permitan representar estados de carga baja, media y alta. Además, los tipos de aplicaciones se agrupan según el tamaño promedio de sus tareas, quedando en el primer grupo los tipos de aplicación 1, 2 y 3, y en el segundo los tipos 4, 5 y 6. Así, se tendrá la siguiente tabla de valores:

Carga	Grupo 1	Grupo 2
Baja	15- 30 min	40 - 60 min
Media	10 - 15 min	25 - 35 min
Alta	4 - 9 min	8 - 20 min

Tabla 5.1: Tiempo entre llegadas de las aplicaciones de cada grupo según la carga

5.2 Evaluación del núcleo administrador del sistema

Las pruebas del núcleo administrador del sistema persiguen, por una parte comprobar la efectividad del esquema propuesto y, por otro lado, comparar las diversas alternativas en los algoritmos de asignación dependiendo de las características de las aplicación y de la plataforma del ambiente. Para cumplir tales objetivos se desarrolló un programa que permite medir las alternativas en dos aspectos fundamentales: el tiempo de ejecución y el volumen de comunicación que no podrá ser transferido directamente. Este programa recibe como parámetros:

Número de tareas de la aplicación.

Tamaño promedio de las tareas.

Porcentaje del tiempo empleado para comunicaciones en cada tarea.

Número de enlaces promedio por tarea

Un archivo con la descripción de la plataforma: grafo de la topología, D_{max}, circuito Hamiltoniano (si lo tiene), etc.

Para realizar las mediciones se tomaron los rangos de valores para las aplicaciones descritos en la sección 5.1; agregándose aplicaciones con un número de tareas superior

a 100. En cuanto a las plataformas de prueba se seleccionaron las siguientes (por ser las mas comunes y las de mayor uso):

- Hipercubos de dimensión 2, dimensión 3 y dimensión 4 (Figura 5.1).
- Mallas: de 8 y de 16 procesadores (Figura 5.2) (la de 4 es un hipercubo de dimensión 2)

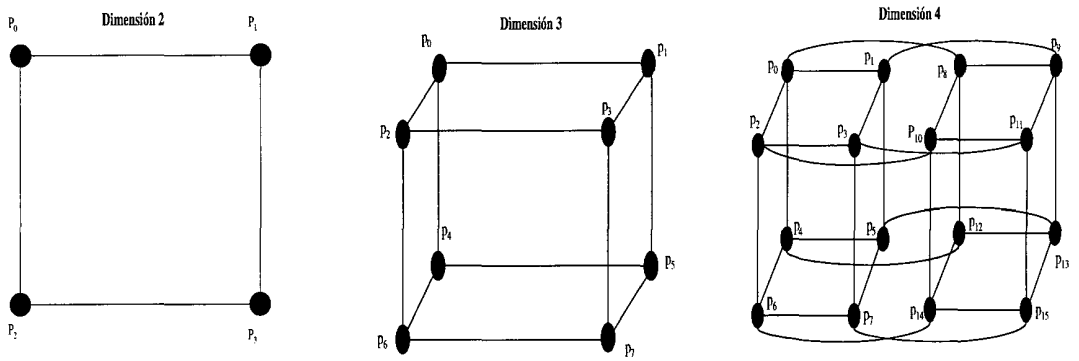


Figura 5.1: Arquitecturas Hipercúbicas

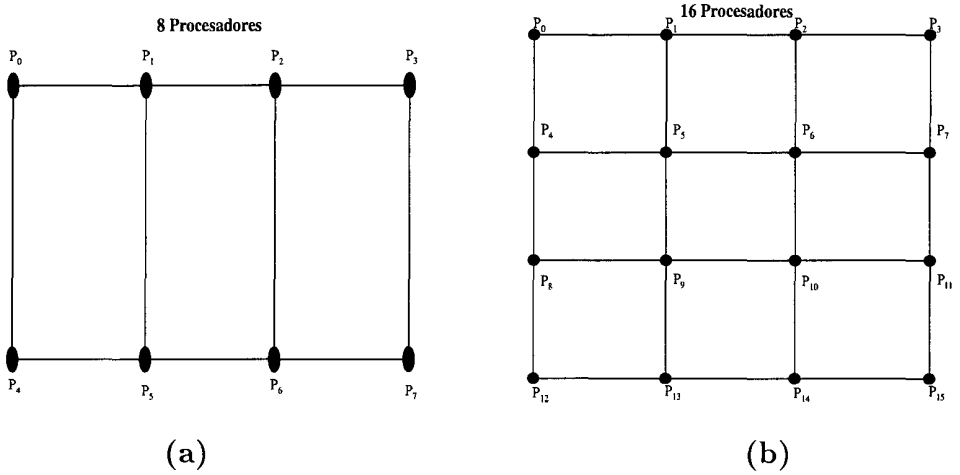


Figura 5.2: Arquitecturas tipo malla

Además, se generaron arquitecturas aleatorias de 4, 8 y 16 procesadores para medir, de alguna forma, el comportamiento de los algoritmos para estructuras desconocidas. Las Figuras 5.3 y 5.4 muestran las topologías utilizadas.

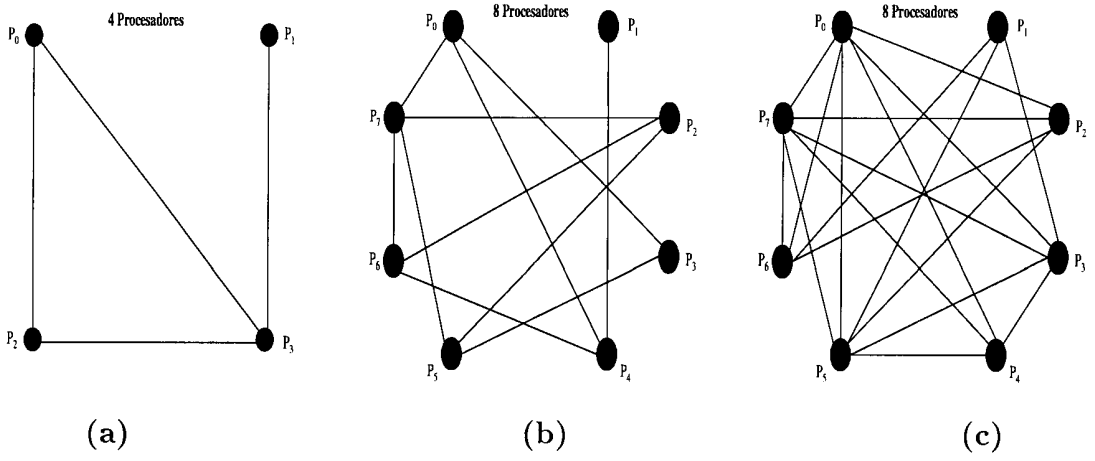


Figura 5.3: Arquitecturas aleatorias de 4 y 8 procesadores

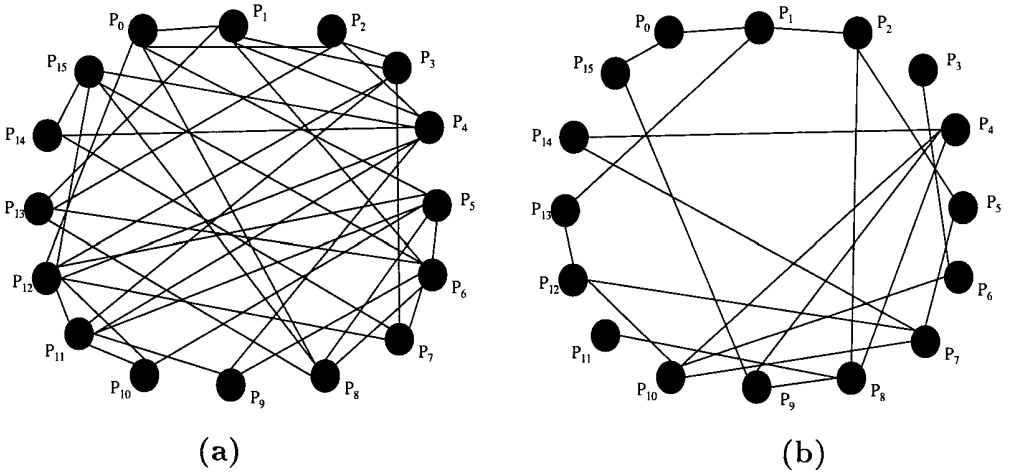


Figura 5.4: Arquitecturas aleatorias de 16 procesadores

5.2.1 Resultados y Análisis

La Figura 5.5 muestra los tiempos de duración de las alternativas para redes aleatorias (con ciclo hamiltoniano) cuyas topologías se muestran en las Figuras 5.1(a) (en este caso, se usa un hipercubo de dimensión 2 como arquitectura de 4 procesadores con ciclo hamiltoniano), 5.3(c) y 5.4(a).

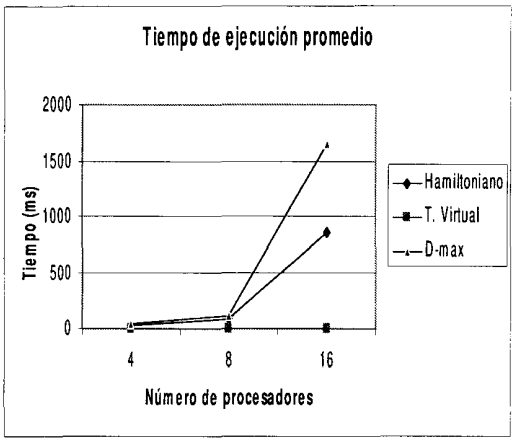


Figura 5.5: Tiempo de ejecución vs Número de procesos en topologías aleatorias

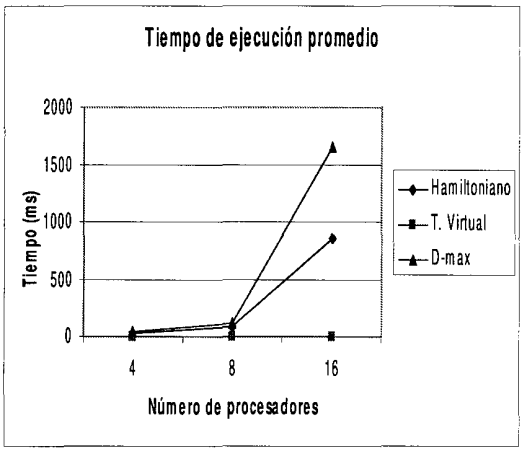
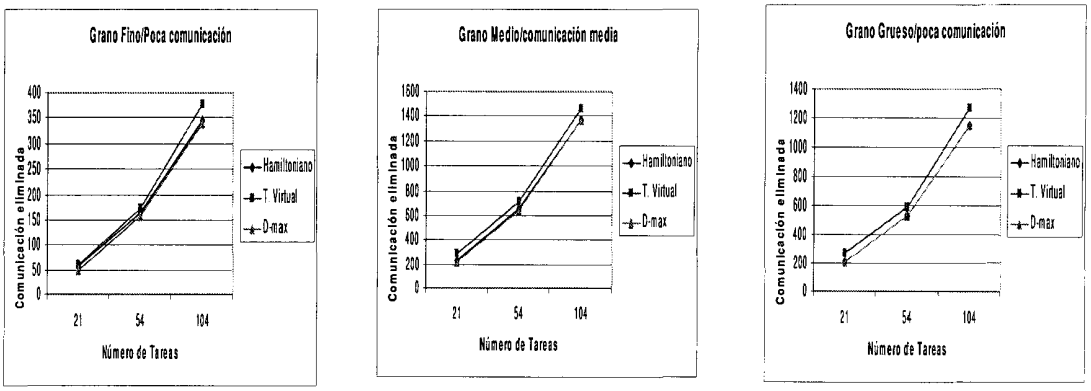


Figura 5.6: Tiempo de ejecución vs Número de procesos en topologías hipercúbicas

De las Figuras 5.5 y 5.6 se observa que el tiempo de duración de la alternativa basada en la topología virtual es siempre cero (0), lo que era de esperarse, puesto que dicha alternativa no involucra un procesamiento complejo. En cuanto a las alternativas que

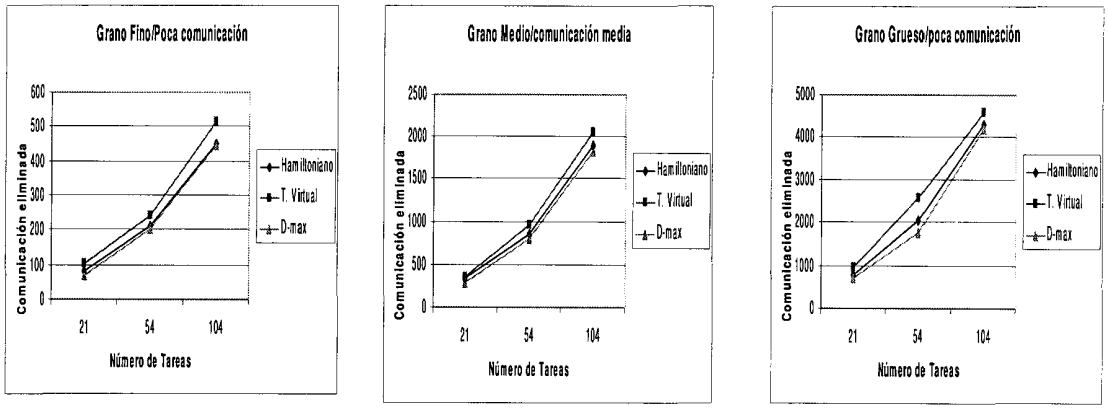
requieren soluciones a problemas NP-completos, se observa que la de D-acoplamiento, tiene en todos los casos una mayor duración que la basada en circuito hamiltoniano máximo. La diferencia entre los tiempos de estas dos alternativas se debe, fundamentalmente, a que la de D-acoplamiento utiliza una función de evaluación más compleja que la de circuito hamiltoniano. En cuanto al orden de magnitud de los tiempos, se presenta el incremento esperado de los mismos a medida que aumenta el número de procesadores (dimensión del problema).

Para las observaciones referentes al volumen de comunicación que es eliminado en cada alternativa (por lo cual deberá ser enrutado), se presentan a continuación las gráficas para tres tipos de aplicaciones.



(a) (b) (c)

Figura 5.7: Volumen eliminado vs Número de tareas para hipercubo de dimensión 3

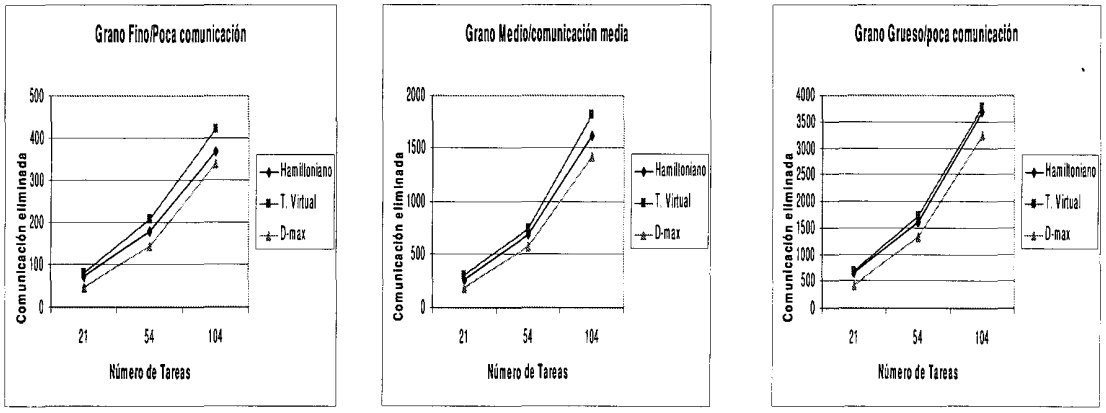


(a) (b) (c)

Figura 5.8: Volumen eliminado vs Número de tareas para malla de 16 procesadores

Las Figuras 5.7 y 5.8 muestran que las alternativas de circuito hamiltoniano máximo

y máximo D-acoplamiento siempre obtienen un mejor resultado que la de topología virtual. La deficiencia, en cuanto a resultados, de la topología virtual se debe a que este esquema trata de minimizar el volumen eliminado en cada paso sin tomar en cuenta el volumen global, cosa que si hacen los otros dos esquemas. En las Figuras 5.7 y 5.8, no se aprecia una diferencia significativa entre D-acoplamiento y circuito hamiltoniano. Esta diferencia se puede observar en la tabla 5.2 y en la Figura 5.9 que muestran que con máximo D-acoplamiento siempre se obtiene un resultado igual o mejor al de circuito hamiltoniano máximo. Por lo tanto, D-acoplamiento aventaja a la alternativa de circuito hamiltoniano, puesto que da tan buenos, o mejores resultados que esta última, y puede utilizarse aún cuando la plataforma no tenga un circuito hamiltoniano. Los mejores resultados de D-acoplamiento se deben a que esta alternativa trata de minimizar el volumen eliminado globalmente; mientras que con circuito hamiltoniano se consigue un circuito máximo que no toma en cuenta los enlaces que no pertenecen a dicho circuito.



(a) (b) (c)

Figura 5.9: Volumen eliminado vs Número de tareas para redes aleatorias de 16 procesadores

	GF/PC			GM/MC			GG/PC		
Alternativa	4	8	16	4	8	16	4	8	16
Cir. Hamiltoniano	160	314	468	638	1246	1913	551	1026	1478
D-acoplamiento	160	306	394	638	1222	1655	551	1014	1337

GF: Grano fino, **GM**: Grano medio, **GG**: Grano Grueso
PC: Poca comunicación, **MC**: Comunicación media

Tabla 5.2: Comunicación eliminada por número de procesadores

En las todas las arquitecturas de 4 procesadores la alternativa basada en ciclo hamiltoniano máximo y la basada en d-acoplamiento siempre emiten los mismos resultados. Para estos casos, la dimensión del problema hace que la solución por ambas alternativas sea la misma, ya que el número de enlaces es bajo.

5.3 Evaluación de la arquitectura del SMA basado en los sistemas multiagentes

En esta parte se describe el simulador de eventos discretos para evaluar el comportamiento del SMA según diferentes configuraciones en la arquitectura del mismo. Las actividades de los agentes no se programan (se asumen tiempos de duración para hacer sus tareas similares, salvo en algunas excepciones), ya que lo que se desea es encontrar la influencia de la arquitectura en el comportamiento del SMA y en su rendimiento global. A continuación se describen los eventos que manejará el simulador:

1. Generación de una aplicación
   (a) La aplicación es recibida por el agente decisorio. (Duración t_1)
   (b) El agente decisorio analiza los parámetros de la aplicación y la clasifica según su paradigma de programación paralela. (Duración: t_2)
   (c) Análisis, por parte del agente decisorio, del estado global del sistema. (Duración: t_3)
   (d) Determinación del esquema de planificación según el tipo de aplicación y el estado del sistema. (Duración: t_4)
   (e) El agente decisorio solicita la planificación al agente planificador adecuado. El tiempo de esta actividad esta determinado, principalmente, por el tiempo de envio de la aplicación. (Duración: t_5)
   (f) Recepción de solicitud por el agente planificador.
   (g) Si el agente planificador usa verificación de las restricciones en el sistema:
      i. Obtención del grafo de ejecución de acuerdo al número de tareas y requerimientos comunicacionales. Esta actividad tiene un tiempo directamente proporcional al tamaño de la aplicación (número de tareas) y al número de agentes ejecutores disponibles. (Duración: t_6)
      ii. Verificación de las restricciones. El agente planificador realiza modificaciones al grafo de ejecución para que este se acople a la topología de interconexión de los ejecutores. El tiempo de duración de este proceso depende del grafo de ejecución resultante y de la topología de interconexión de los ejecutores, asumiendo tiempos similares para ambas alternativas. (Duración: t_7)
   (h) Si el agente planificador usa topología virtual
      i. Obtención del grafo de ejecución de acuerdo al número de tareas y requerimientos comunicacionales. (Duración: t_6)
      ii. Establecimiento de enlaces virtuales entre los agentes ejecutores para generar la topología virtual. Esta actividad tendrá un tiempo de duración proporcional al número de enlaces virtuales que se necesitan. (Duración: t_8)

(i) Generación de una nueva llegada para ese tipo de aplicación. (Agente Decisorio).

(j) Generación del evento de asignación de la carga.

2. Asignación de carga por parte del agente planificador.

   (a) Revisión del estado de los agentes ejecutores. (Duración: t_9)

   (b) Asignación de las tareas a los ejecutores según sus capacidades y su estado real. (Duración: t_{10})

   (c) Generación del evento de creación de las tareas.

3. Creación de tareas. (Agente ejecutor)

   (a) El agente ejecutor crea las tareas que le han sido asignadas colocándoles un tiempo estimado de finalización. (Duración: t_{11})

   (b) Generación del evento de culminación de tarea.

4. Culminación de tareas.

   (a) El agente ejecutor notifica al planificador la finalización de una tarea. (Duración: t_{12})

   (b) El agente planificador registra el fin de una tarea que pertenece a una determinada aplicación. (Duración: t_{13})

   (c) El agente planificador determina si la aplicación culmino su ejecución. Si es cierto, genera un evento de finalización de la aplicación.

5. Finalización de aplicación

   (a) El planificador reporta fin de aplicación.

   (b) El agente decisorio almacena estadisticas de la ejecución de dicha aplicación.

5.3.1 Resultados y análisis

Las pruebas sobre el simulador se realizaron tomando tres esquemas de diseño:

- Esquema Centralizado (**EC**): Para este caso se utiliza un único agente decisorio, un agente planificador por cada tipo de planificación, y tantos agentes ejecutores como procesadores existan en el sistema.
- Esquema de distribución total (**EDT**): Se tienen tantos agentes decisorios y agentes por tipo de planificación como agentes ejecutores tenga el sistema.
- Esquema de distribución parcial (**EDP**): Para este caso se estudian diferentes alternativas, teniendo agentes decisorios y agentes por tipo de planificación para grupos de agentes ejecutores.

EDP-N: N agentes decisorios y N agentes por tipo de planificación para todo el conjunto de agentes ejecutores.

Las pruebas se realizaron con el objeto de comparar los esquemas para 4, 8 y 16 procesadores (agentes ejecutores), tomando las tasas de entrada mostradas en la tabla 5.1 y usando los valores de t_i, en cada caso, según la tabla 5.3. Se toma un valor de 3 para todos los tiempos de preparación de tareas de los diferentes agentes (t_1 .. t_6 y t_{11}.. t_{13}), para garantizar que la actividad tendrá un tiempo menor al tiempo más corto entre llegadas de aplicaciones. Además, se colocan valores distintos para t_7 y t_8 puesto que, estas representan actividades que consumen más tiempo que las anteriores. Por otro lado, t_{10} toma un valor distinto dependiendo del tipo de planificación usada, ya que esta refleja el proceso de asignación de las tareas.

Tiempo	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}
Valor	3	3	3	3	3	3	9	6	3	5,10	3	3	3

Tabla 5.3: Tiempos de duración de los eventos

En cada caso se tomó:

- **MA**: Tiempo promedio que tarda el sistema en arrancar una aplicación. Es decir, tiempo promedio desde que la aplicación llega al sistema hasta que comienza a ejecutarse.
- **AD**: Tiempo promedio de espera de un evento por ser atendido por algún agente decisorio.
- **AP**: Tiempo promedio de espera de un evento por ser atendido por algún agente planificador.
- **% AD**: Porcentaje de ocupación promedio de los agentes decisorios.
- **% AP**: Porcentaje de ocupación promedio de los agentes planificadores.

Las tablas 5.4, 5.5 y 5.6 muestran que el EC los valores de los tiempos (MA, AD, AP) y de los porcentajes de ocupación (%AD, %AP) se mantienen constantes al variar el número de agentes ejecutores, esto se atribuye al hecho de que existe un único agente decisorio y un agente por tipo de planificación, independientemente del número de agentes ejecutores del sistema. El EC tiene la MA más alta para todos los casos, ya que las aplicaciones deben esperar para ser atendidas por el único agente decisorio del sistema. Por otro lado, los EDP reducen MA y AD sin incrementar significativamente los AP; presentando un porcentaje de uso para los agentes (%AD y %AP) más alto que EDT. Los EDT logran el mínimo MA; pero, presentan el peor uso de los recursos, esto

es, valores más bajos para %AD y %AP. Puede observarse de las mismas tablas, que el %AD siempre es mayor que el %AP y que este último se incrementa al disminuir el primero. Esto implica, que el uso de los agentes planificadores esta limitado por la capacidad de procesamiento del agente decisorio.

Nproc	4					8				
Dato	MA	AD	AP	%AD	%AP	MA	AD	AP	%AD	%AP
EC	2316.4	636.6	0.2	100	35	2316.4	636.6	0.2	100	35
EDT	42.4	0.04	1.53	80	60	35.5	0	0	40	30
EDP-2	1373	360.4	1.86	100	60	1373	360.4	1.86	100	60
EDP-4	No Aplica					42.4	0.04	1.53	80	60
EDP-8	No Aplica					No Aplica				

Nproc	16				
Dato	MA	AD	AP	%AD	%AP
EC	2316.4	636.6	0.2	100	35
EDT	35.5	0	0	20	15
EDP-2	1373	360.4	1.86	100	60
EDP-4	42.4	0.04	1.53	80	60
EDP-8	35.5	0	0	40	30

Tabla 5.4: Resultados para carga alta

Nproc	4					8				
Dato	MA	AD	AP	%AD	%AP	MA	AD	AP	%AD	%AP
EC	2045	540.4	1.9	100	60	2045	540.4	1.9	100	60
EDT	35.5	0	0	40	30	35.5	0	0	20	15
EDP-2	41.7	0.16	1.12	78	60	41.7	0.16	1.12	78	60
EDP-4	No Aplica					35.5	0	0	40	30
EDP-8	No Aplica					No Aplica				

Nproc	16				
Dato	MA	AD	AP	%AD	%AP
EC	2045	540.4	1.9	100	60
EDT	35.5	0	0	10	7.5
EDP-2	41.7	0.16	1.12	78	60
EDP-4	35.5	0	0	40	30
EDP-8	35.5	0	0	20	15

Tabla 5.5: Resultados para carga media

Si se toma en cuenta que un buen esquema debería tener valores bajos de AD, AP y MA, y altos de %AD y %AP; puede decirse que EDP-2 es el mejor esquema para todos los casos de cargas medias y bajas, mientras que EDP-4 es mejor para cargas altas con mas de 8 procesadores. Por otra parte, EDT es mejor para cargas altas con menos de 8 procesadores.

Nproc	4					8				
Dato	MA	AD	AP	%AD	%AP	MA	AD	AP	%AD	%AP
EC	56.3	1.87	3.7	90	70	56.3	1.87	3.7	90	70
EDT	35.5	0	0	22	17	35.5	0	0	11	8.4
EDP-2	35.6	0	0.04	44	32	35.6	0	0.04	44	32
EDP-4	No Aplica					35.5	0	0	22	16
EDP-8	No Aplica					No Aplica				

Nproc	16				
Dato	MA	AD	AP	%AD	%AP
EC	56.3	1.87	3.7	90	70
EDT	35.5	0	0	5.6	4.2
EDP-2	35.6	0	0.04	44	32
EDP-4	35.5	0	0	22	16
EDP-8	35.5	0	0	11	8.4

Tabla 5.6: Resultados para carga baja

5.4 Aspectos resaltantes de los resultados

Para cada una de las evaluaciones hechas, se presentan a continuación los puntos más resaltantes que se desprenden de los resultados obtenidos.

5.4.1 SMA basado en Multiagentes

- **Relevancia de los agentes:** Los agentes decisorios resultan claves como elementos de toma de decisiones. La información que estos agentes manejan es de gran importancia para un mejor rendimiento del sistema, puesto que permiten que el SMA tenga un alto grado de adaptabilidad y flexibilidad. Por otro lado, los agentes planificadores realizan las actividades relacionas con el núcleo administrador, lo que los convierte en prestadores de un servicio fundamental para el sistema.

- **Influencia de la arquitectura del sistema multiagentes.** En este aspecto, debe resaltarse que un arquitectura dinámica (que pueda cambiar según la carga del sistema, el tipo de aplicaciones, el número de procesadores disponibles, etc.) dará al SMA la posibilidad de tener el mejor rendimiento según las características reales del sistema.

5.4.2 Núcleo administrador del sistema

- **Relación con la arquitectura de la plataforma.** Se conserva el hecho que para todos las arquitecturas la mejor alternativa resulta ser la de D-acoplamiento. En cuanto a la influencia de la arquitectura en el volumen de comunicación elim-

inada, los resultados muestran que a menor número de enlaces en la arquitectura de la plataforma mayor será el volumen de comunicación eliminada.

- **Costos de las alternativas.** sin tomar en cuenta el tiempo de ejecución, una alternativa es mejor mientras menor sea el volumen de comunicación que debe eliminarse; este volumen eliminado implicará un retardo para la ejecución de la aplicación, ya que dicha comunicación deberá ser enrutada por el sistema a través de procesadores distintos a los que están involucrados en la comunicación.

Conclusiones

En este trabajo, se ha presentado un estudio sobre el problema de reconfiguración de plataformas de procesamiento paralelo/distribuido. La propuesta involucra dos aspectos fundamentales, por una parte la resolución de los problemas de particionamiento, asignación y mapeo de procesos, y por otro lado, una visión multiagentes de la solución para el sistema manejador del ambiente.

Existen mecanismos de solución para la multiplicidad de problemas asociados a los ambientes de procesamiento paralelo/distribuido. Los sistemas multiagentes representan un vía de integración efectiva para estos mecanismos, permitiendo darle al sistema un alto grado de adaptabilidad y flexibilidad. Así, se han presentado dos enfoques de diseño, uno en el que la aplicación se adapta al sistema y otro que sigue el procedimiento inverso. Se presentaron varios problemas de tipo NP-completos lo cuales se resolvieron usando Algoritmos Genéticos.

Las pruebas realizadas en el SMA basado en agentes del sistema muestran que el tiempo promedio de espera de los eventos disminuye a medida que se aumenta los tiempos de llegada entre las aplicaciones; sin embargo, se estabiliza cuando las aplicaciones llegan muy espaciadas. Las observaciones interesantes en este punto tienen que ver con las posibilidades de modificación de la arquitectura del sistema multiagentes que le permiten adaptarse a las condiciones de operación de la plataforma. Tal como muestran los resultados, un esquema de distribución parcial con dos agentes decisorios es mejor en los casos de tasas de llegadas medias y bajas; mientras que, la distribución parcial con 4 agentes decisorios es la de mejor rendimiento para plataformas de más de 8 procesadores con tasas de llegadas altas.

En cuanto a la evaluación del núcleo administrador del sistema, es claro, según los resultados, que la mejor alternativa es la basada en el algoritmo de máximo D-acoplamiento, puesto que resuelve los problemas de asignación con el menor costo; sin requerir que la plataforma y el grafo de agrupamiento tengan algún circuito hamiltoniano. Sin embargo, debe destacarse el incremento que tiene el tiempo de ejecución de esta alternativa, el cual podría ser significativo para plataformas de muchos procesadores; en cuyo caso puede establecerse un compromiso entre el tiempo de duración del algoritmo y la calidad de la solución.

Para finalizar, la construcción del simulador plantea una amplia gama de posibilidades de estudio y análisis de alternativas. Entre estas alternativas se pueden señalar: incorporación de capacidades tanto a nivel individual de los agentes como a nivel colec-

tivo, otros esquemas de resolución de los problemas de agrupamiento y asignaciones que puedan derivarse de nuevas teorías o hibridaciones de las existentes. Además, se abre un interesante camino hacia la afinación e implementación de Sistemas Multiagentes para manejo de recursos en ambientes de procesamiento paralelo/distribuido el cual puede ser explorado a través de prototipos que trabajen en ambientes de operación real.

Referencias

[1] Toru ISHIDA. *Parallel, Distributed and Multiagent Production Systems.* Springer-Verlag, Berlin, Germany, 1994.

[2] B. CHIAD-DRAA and LEVESQUE P. *Hierachical Model and Communications by Signs, Signals and Symbols in Multiagent Enviroments.* Saint-Foy, Canada, 1995.

[3] C. BOUTILIER, Y. SHOHAM, and M.P. WELLMAN. Economic principies of multi-agent systems. *Artificial Intelligence*, 94(1).

[4] S. KRAUS. Negotation and cooperation in multi-agent enviroments. *Artificial Intelligence*, 94(1).

[5] M. WOOLDRIDGE and N.R. JENNINGS. Intelligent Agents: Theory and Practice . *Knowledge Engineering Review.*

[6] A. CHAVEZ, A. MOUKAS, and P. MAES. *Challenger: A Multi-agent System for Distributed Resource Allocation.* MIT Media Lab., Cambridge, MA, 1996.

[7] Deo NARSINGH. *Graph Theory with Applications to Engineering and Computer Science.* Prentice-Hall, INC., Englewood Cliffs, N.J., U.S.A., 1974.

[8] L.R. FOULDS. *Combinatorial Optimization for Ungraduates.* Springer-Verlag, New York, USA, 1984.

[9] Robert SEDGEWICK. *An Introduction to the Analysis of Algorithms.* Addison-Wesley., 1996.

[10] John HOLLAND. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[11] D. GOLBERG. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley., 1989.

[12] L. DAVIS. *Handbook of Genetic Algorithms.* Strand Reinhold, New York, USA, 1991.

[13] M. SRINAS and M. PATNAIK. Genetic Algorithms: A Survey. *IEEE Computer*, pages 17–26, 1994.

[14] J. FERBER. Reactive distributed artificial intelligence: Principles and applications. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence.* John Wiley & Sons, 1996.

[15] Jose AGUILAR. *L'Allocation de Táches, l'Equilibrage de Charge et l'Optimisation Combinatoire.* Phd thesis, Université René Descartes - Paris V, Paris, France, 1995.

[16] Francisco HIDROBO and Jose Aguilar. Algoritmos genéticos paralelos en problemas de optimización combinatoria. *REVISTA TECNICA DE LA FACULTAD DE INGENIERIA DE LA UNIVERSIDAD DEL ZULIA*, 21(1):47–58, 1998.

[17] Francisco HIDROBO. *Esquemas de Paralelización y Reforzamiento para Algoritmos Genéticos Aplicados a Problemas de Optimización Combinatoria.* Trabajo de ascenso, Universidad de Los Andes, Mérida, Venezuela, 1997.