# IMPLEMENTACIÓN DE UN CLÚSTER DE CÁLCULO DISTRIBUIDO PARA EL PROGRAMA DE INGENIERÍA ELECTRÓNICA

JAIME ANDRÉS ROMERO LEIVA
FERNANDO ANDRÉS SILVA ORTIZ

UNIVERSIDAD SURCOLOMBIANA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
NEIVA
2007

# IMPLEMENTACIÓN DE UN CLÚSTER DE CÁLCULO DISTRIBUIDO PARA EL PROGRAMA DE INGENIERÍA ELECTRÓNICA

JAIME ANDRÉS ROMERO LEIVA
FERNANDO ANDRÉS SILVA ORTIZ

Trabajo de grado presentado como requisito para
Optar al título de Ingeniero Electrónico

Director
JOSÉ DE JESÚS SALGADO PATRÓN
Magíster en Ingeniería Electrónica

UNIVERSIDAD SURCOLOMBIANA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
NEIVA
2007

**Nota de aceptación:**

_____

_____

_____

_____

_____

_____

**Firma del presidente del jurado**

_____

**Firma del jurado**

_____

**Firma del jurado**

Neiva, 29 de octubre de 2007

A Dios, a mi familia, a mis padres,
Que con su trabajo y dedicación,
Depositaron en mi toda su confianza
Y me apoyaron hasta el final,
Haciendo realidad mis sueños.
**Jaime Andrés**

A mi familia, que con su apoyo incondicional
Han hecho posible culminar con éxito esta etapa en mi vida;
Y a mis amigos, que con su ayuda y compañía hicieron
De estos años una época imborrable
**Fernando Andrés**

## AGRADECIMIENTOS

Los autores expresan sus agradecimientos a:

A Dios

A nuestras familias

Al ingeniero José de Jesús Salgado Patrón, director de Tesis.

Al Ingeniero Jorge Antonio Polanía Puentes.

Al Ingeniero Yamil Armando Cerquera Rojas.

Al Ingeniero Diego Jiménez Terranova.

Al ingeniero Agustín Soto Otalora.

A todos los docentes, que han colaborado con nuestra educación.

A nuestros compañeros y amigos, en especial a:

Hamilton Rivera Perez

Juan Gilberto García Lopez

Leidy Diana Rondón Bautista

Mario Fernando González Valencia

Y a todos aquellos que de una u otra forma colaboraron con la realización de éste proyecto.

**CONTENIDO**

## LISTA DE TABLAS

Pág

## LISTA DE FIGURAS

## LISTA DE ANEXOS

## *RESUMEN*

El cálculo distribuido no es un procedimiento nuevo en cuanto a solución de problemas de alto costo computacional se refiere, de hecho nace con la invención del computador moderno, pues siempre existen problemas que requieren más capacidad de procesamiento y velocidad que el solucionado por un PC convencional; además, mientras más avanza la tecnología, surgen mayores problemas computacionales por resolver.

La funcionalidad de un sistema distribuido es la solución a un gran problema para lo cual requiere la utilización de un conjunto de computadores conectados bajo una red debidamente estructurada, dando la sensación que dicha solución se obtiene a partir de un solo ordenador y otorga una serie de ventajas como lo son la concurrencia, alta escalabilidad, compartir recursos y tolerancia a fallos; y desventajas como complejidad al software y disminución de los niveles de seguridad, que se ven opacadas ante la buena relación precio/funcionalidad presentada por este sistema y la capacidad de aumentar su tamaño si la carga de trabajo lo requiere.

La esencia de un clúster, es un ordenador con requerimientos mínimos de hardware, con los que se pueda conectar a una red específica. En la red de cálculo distribuido, el ordenador se convertirá en la unidad básica de procesamiento conocida como nodo. Con el fin de obtener una eficiencia óptima o aceptable del sistema, se necesita la utilización de varios ordenadores individuales que, trabajando en paralelo, se conseguirá una capacidad de cálculo respetable. La función del paralelismo es el reducir el número de ciclos de ejecución de un programa, en relación al número de procesadores que existen en el sistema.

La idea principal de este proyecto es la implementación de un cluster con los requerimientos anteriormente mencionados para el programa de Ingeniería Electrónica teniendo en cuenta los recursos disponibles de la sala de simulación, con el objetivo de desarrollar nuevas tecnologías y metodologías de avance, que resalten a nivel regional los estudios realizados por dicho programa. Para la ejecución de este proyecto, se utiliza el MATLAB® versión 7.4, debido a que es una herramienta potente para el desarrollo de software orientado a Ingeniería.

## *ABSTRACT*

*Distributed calculation is not a new procedure for problems solution of high computational cost it refers, in fact it's born with the invention of the modern computer, because always exist problems that require more prosecution speed and capacity that solved by a conventional PC; also, while technology advances, bigger computational problems arise to solve.*

*Functionality of a distributed system is the solution to a great problem for which it requires the use of a set of computers connected under a properly structured network, giving the sensation that this solution is obtained from a single computer and grants a series of advantages as they are the concurrence, high scalability, sharing resources and tolerance to failures; and disadvantages like software complexity and diminution of the security levels, that are reduced by the good price/functionality rate given by this system and the capacity to increase their size if the service load requires it.*

*Cluster's essence is a computer with minimum hardware requirements, with which it is possible to be connected to a specific network. In the Distributed calculation network, the computer will become the processes basic unit known like node. With the purpose of obtaining an optimal or acceptable efficiency of the system, it's needed the use of individual computers that, working in parallel, will obtain a respectable capacity of calculation.*

*The main idea of this project is the implementation of a cluster with the requirements previously described for the electronic engineering program taking into account the available resources of the simulation room, with the objective of developing new technologies and progress methodologies, which shows a regional studies by the program. For the implementation of this project, using the MATLAB ® Version 7.4, because it is a powerful tool for engineering software development.*

# INTRODUCCIÓN

*La ingeniería es la rama del conocimiento humano más dinámica, de permanente innovación y renovación. Siempre ha buscado resolver problemas de mayor envergadura, cada vez en menos tiempo y presentando soluciones más optimizadas en todos los aspectos (potencia, costo, escalabilidad, etc.).*

*El cálculo distribuido ha contribuido a la solución de complejos problemas en diversas áreas del conocimiento; por citar algunas: astronomía, ingeniería, física, matemáticas, paleontología, criminalística y medicina forense. En la actualidad, el área investigativa y científica ha dejado de emplear una única supercomputadora de elevado costo, pasando a implementar redes de computadoras, mucho más económicas, trabajando en paralelo y distribuyendo problemas complejos en tareas más sencillas. El objetivo principal del paralelismo es el reducir el número de ciclos de ejecución de un programa en relación al número de procesadores que existen en el sistema.*

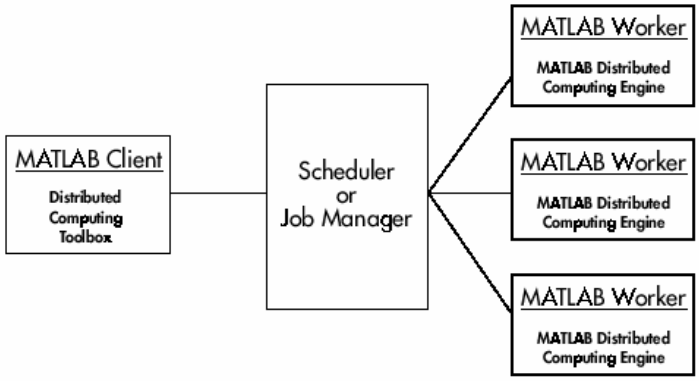
*Se evidencia entonces una clara necesidad: implementar un sistema de este tipo en nuestra universidad, con el fin de facilitar el trabajo investigativo, la generación de nuevo conocimiento y la admisión de la programación paralela como práctica de programación en nuestro pensum; todo esto buscando ubicar a la Universidad Surcolombiana a la vanguardia investigativa a nivel regional, y a un nivel competitivo en esta área a nivel nacional.*

# 1. DESCRIPCIÓN GENERAL DEL SISTEMA

Debido a que muchas entidades de investigación y afines del país emplean computadores de alta gama (ordenadores de alto costo) comúnmente llamados supercomputadores para realizar diversas tareas de investigación a gran velocidad y que son fabricadas por compañías extranjeras reconocidas a un alto precio, se hace necesario implementar un clúster de cálculo distribuido el cual cumpla con las exigencias propias de nuestro entorno. Estas exigencias constan de: Rapidez para solucionar problemas computacionales grandes, rendimiento, escalabilidad, bajo precio y alta confiabilidad.

Este proyecto consiste en implementar un clúster de cálculo distribuido bajo plataforma MatLab que utilice el procesador de cada computador ubicado en la sala de simulación del programa de Ingeniería Electrónica, uniéndolos de manera virtual en una subred predeterminada para formar un sistema de computación paralela y de esta forma realizar diversas tareas a una gran velocidad. El sistema consta de un computador cliente el cual se encarga de asignar el programa segmentado en trabajos, pasarlos a un Job Manager (Administrador de trabajo), quien a su vez asigna las tareas a cada ordenador (worker).

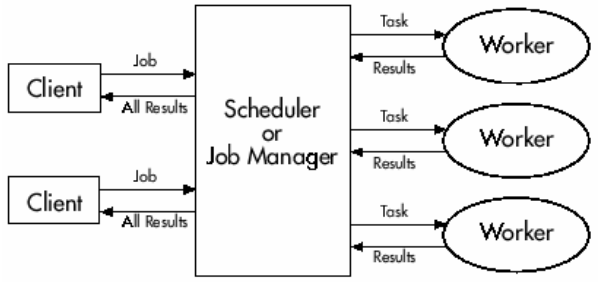
Figura 1. Diagrama general del clúster




Fuente: Ayuda de Mathworks MATLAB 7.4

La integración del clúster con MatLab 7.4 es innata, ya que la versión completa del software (MatLab versión 7.4 licenciado por Mathworks) contiene un toolbox dedicada a este fin, que facilita la realización del proyecto. El servicio principal tiene por nombre MATLAB Distributed Computing Engine (MDCE). Bajo programación se puede implementar diferentes configuraciones de clúster dentro del mismo sistema como lo muestran las siguientes figuras:

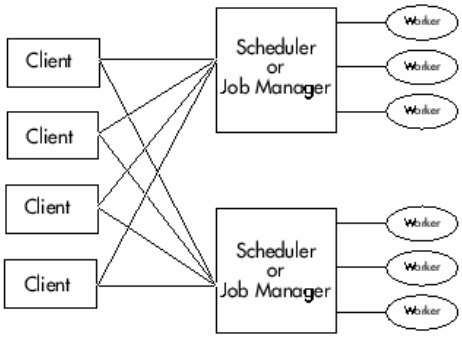
Figura 2. Interacciones de cálculo distribuido con más de una sesión (cliente)




Fuente: Ayuda de Mathworks MATLAB 7.4

En esta configuración el cluster contiene un solo Job Manager y éste se encuentra registrado con sus respectivos workers

Figura 3. Configuraciones con múltiples clientes y Job Manager




Fuente: Ayuda de Mathworks MATLAB 7.4

En esta configuración, para una red grande puede incluir varios Job Managers y varias sesiones cliente. Cualquier sesión cliente puede crear, correr y acceder trabajos en cualquier Job Manager, pero una sesión worker está registrada y dedicada a sólo un Job Manager a la vez.

# 2. OBJETIVOS

## Objetivo General

- Implementar un sistema de cómputo distribuido en la sala de simulación del programa de Ingeniería electrónica de la Universidad Surcolombiana que sirva de herramienta de cálculo en aplicaciones que requieran un procesamiento robusto, rápido y confiable.

## Objetivos Específicos

- Implementar una plataforma de cálculo distribuido para simulaciones de tipo investigativo y académico.
- Incitar la creación de nuevos grupos de investigación ofreciéndoles una valiosa herramienta que acelere la solución de problemas de alto costo computacional.
- Fomentar en los directivos, docentes y estudiantes el interés por la programación paralela y el cálculo distribuido, a través de la creación de algunas prácticas que involucren al programa en estas aplicaciones tecnológicas.
- Demostrar experimentalmente las ventajas del método propuesto con respecto a la utilización de un solo ordenador.

Para alcanzar los objetivos citados, la metodología de trabajo se basó en los siguientes puntos:

- Análisis de las alternativas de implementación disponibles.
- Profundización en el conocimiento de la plataforma elegida (MATLAB).
- Implementación directa del clúster mediante comandos de consola.
- Desarrollo de un software en MATLAB que permita administrar el clúster de forma sencilla sin requerir un conocimiento profundo de la consola de instalación (DOS) y sus comandos.
- Evaluación experimental del rendimiento del clúster mediante la ejecución de aplicaciones computacionalmente exigentes.
- Comparación con los resultados obtenidos al utilizar programación convencional en un solo nodo.

# 3. MARCO TEÓRICO

## 3.1 PROCESAMIENTO PARALELO Y PROCESAMIENTO DISTRIBUIDO*

El procesamiento paralelo consiste en resolver *simultáneamente* varias tareas independientes que conforman un problema de gran dimensión o complejidad. Esta tecnología ha emergido como una importante opción en la computación moderna de alto rendimiento, siendo los últimos años testigos de una aceptación y adopción creciente del procesamiento paralelo, tanto para computación científica de alto desempeño como para propósitos más generales como la administración y consulta de grandes bases de datos.

El procesamiento distribuido de datos es aquel en el cual la cantidad de información o tareas a manejar es muy grande y para hacer cálculos con el de forma rápida se segmenta esta información, de forma que los nodos del sistema hagan la misma tarea pero sobre diferentes áreas o porciones de la información.

En consecuencia a estas proyecciones y tendencias, estas plataformas de trabajo están captando cada vez más el interés del mercado y la atención de los investigadores como una interesante opción que permite el procesamiento de grandes volúmenes de información y la resolución de problemas computacionales de gran complejidad, que de ser resueltos en un solo procesador exigirían mucho más tiempo.

La plataforma computacional necesaria para la aplicación del procesamiento paralelo consiste en un sistema computacional distribuido/paralelo, es decir, un conjunto de procesadores interconectados entre sí por una red de comunicación, pudiendo utilizarse para este fin inclusive microprocesadores de muy bajo costo. Es evidente las ventajas económicas que acarrea esta implementación, teniendo en cuenta el gran parque de computadores personales (PC) disponibles en la actualidad.

Las técnicas de procesamiento paralelo han sido aplicadas con éxito en diversas áreas de la ciencia y la ingeniería, como:

- Astronomía
- Hidrodinámica computacional
- Programas de sismografía
- Óptica física
- Ingeniería genética
- Medicina: tomografía cerebral computada

* tomado de la toolbox de MATLAB

- Diseño de motores
- Mecánica cuántica

La mayoría de estas aplicaciones, así como también las aplicaciones relacionadas con la Ingeniería Electrónica, utilizan modelos matemáticos que implican el procesamiento de gran cantidad de datos. En este contexto, la aplicación del procesamiento paralelo a la resolución de problemas tales como redes neuronales y lógica difusa presenta excelentes perspectivas, como puede apreciarse en trabajos recientes de investigación realizados sobre las áreas anteriormente mencionadas.

## 3.2 SEGMENTACIÓN DEL PROBLEMA

Para que un problema particular sea resuelto utilizando procesamiento paralelo o distribuido, es necesario que dicho problema sea "segmentable", es decir, que las características de los métodos implementados en la resolución del mismo permitan la distribución de diversas tareas a los procesadores componentes del sistema distribuido, de manera tal a resolver el problema en conjunto.

De los numerosos problemas paralelizables que se conocen, ocuparemos preferentemente de los problemas de ingeniería que se plantean como un sistema (no necesariamente lineal) de ecuaciones, que por lo general son resueltos utilizando métodos iterativos. Para paralelizar estos métodos, nacen los métodos bloque-iterativos, que consisten en asignar a los procesadores del sistema distribuido distintos grupos de ecuaciones a ser resueltos localmente. Los resultados obtenidos localmente por cada uno de ellos son transmitidos a los demás a través de la red de comunicación, avanzando el conjunto hacia la solución global del sistema de una forma típicamente iterativa.

La utilización de los métodos de solución bloque-iterativos aplicados a los problemas electrónicos arriba citados en un ambiente computacional paralelo merece especial atención, ya que ha presentado resultados promisorios. Para que la implementación de estos métodos sea realmente eficaz, es importante descomponer la tarea general en pequeñas tareas, con el objetivo de promover el adecuado tratamiento del problema para la arquitectura paralela; esto es, debe descomponerse el problema a resolver en subtareas de tal forma que la implementación del algoritmo de resolución sea computacionalmente eficiente.

La computación paralela desde sus inicios presenta diversos trabajos sobre métodos de partición que descomponen la tarea general utilizando diversos criterios. Teniendo claro el problema se debe descomponer en tareas, que permitan su resolución eficiente utilizando procesamiento paralelo en un sistema distribuido homogéneo (computadores

con características similares). Una resolución eficiente es aquella que utiliza un menor tiempo computacional en llegar a la solución, buscando balancear de manera adecuada la carga computacional a ser repartida a los diversos procesadores y minimizar la cantidad de envíos de mensajes entre procesos paralelos.

De esto último se puede concluir que, considerando la posibilidad de operar en un ambiente computacional compuesto por procesadores de igual desarrollo, interconectados por un sistema de comunicación o subred que la conforme, la descomposición de la tarea estará condicionada por dos factores:

a) El balanceamiento de carga computacional: Este factor determina las dimensiones de las tareas asignadas a cada procesador, que deberán estar en proporción con la capacidad relativa de procesamiento de cada uno de los procesadores.

b) El grado de acoplamiento que puedan tener las tareas entre sí, lo que determina la dependencia entre las variables de las sub-tareas y por consiguiente, influye en la convergencia del algoritmo.

En conclusión, el problema que se plantea, dado un sistema distribuido con cierto número de procesadores, consiste en descomponer una tarea de gran envergadura en varias sub-tareas menores, de forma tal que a cada procesador se le asigne una sub-tarea de dimensión proporcional a su procesador, y que la dependencia entre las variables actualizadas por cada uno de los procesadores facilite la convergencia del algoritmo a ser utilizado.

## 3.3 CLÚSTER

*El término clúster se aplica a los conjuntos o conglomerados de computadoras construidos mediante la utilización de componentes de hardware comunes y que se comportan como si fuesen una única computadora. Juegan hoy en día un papel importante en la solución de problemas de las ciencias, las ingenierías y del comercio moderno.

La tecnología de clústeres ha evolucionado en apoyo de actividades que van desde aplicaciones de súper-cómputo y software de misiones críticas, servidores Web y comercio electrónico, hasta bases de datos de alto rendimiento, entre otros usos.

* Tomado del portal de Internet: www.wikipedia.es.

El cómputo con clústeres surge como resultado de la convergencia de varias tendencias actuales que incluyen la disponibilidad de microprocesadores económicos de alto rendimiento y redes de alta velocidad, el desarrollo de herramientas de software para cómputo distribuido de alto rendimiento, así como la creciente necesidad de potencia computacional para aplicaciones que la requieran.

De un clúster se espera que presente combinaciones de los siguientes servicios:

- Alto rendimiento (High Performance)
- Alta disponibilidad (High Availability)
- Equilibrio de carga (Load Balancing)
- Escalabilidad (Scalability)

Los ordenadores del clúster pueden tener todos, la misma configuración de hardware y sistema operativo (clúster homogéneo), diferente rendimiento pero con arquitecturas y sistemas operativos similares (clúster semi-homogéneo), o tener diferente hardware y sistema operativo (clúster heterogéneo), lo que hace más fácil y económica su construcción.

Para que un clúster funcione como tal, no basta solo con conectar entre sí los ordenadores, sino que es necesario proveer un sistema de manejo del clúster, el cual se encargue de interactuar con el usuario y los procesos que corren en él, para optimizar el funcionamiento.

## 3.4 COMPONENTES DE UN CLÚSTER

En general, un clúster necesita de varios componentes de software y hardware para poder funcionar. A saber*:

- Nodos: Pueden ser simples ordenadores, sistemas multi-procesador o estaciones de trabajo (*workstations*).

- Sistema Operativo: Debe ser de fácil uso y acceso y permitir además múltiples procesos y usuarios. Ejemplos:

✓ GNU/Linux
✓ Unix: Solaris / HP-Ux / Aix

* Tomado del portal de Internet: www.wikipedia.es.

✓ Windows NT / 2000 / 2003 Server / XP
✓ Mac OS X
✓ Clúster OS's especiales

• **Conexiones de Red:** Los nodos de un clúster pueden conectarse mediante una simple red Ethernet con placas comunes (network adapters o NIC'S) , o utilizarse tecnologías especiales de alta velocidad como Fast Ethernet, Gigabit Ethernet, Myrinet, Infiniband, SCI, etc.

• **Middleware:** El middleware es un software que generalmente actúa entre el sistema operativo y las aplicaciones con la finalidad de proveer a un clúster lo siguiente:

✓ una interfaz única de acceso al sistema, denominada SSI (*Single System Image*), la cual genera la sensación al usuario de que utiliza un único ordenador muy potente;
✓ herramientas para la optimización y mantenimiento del sistema: migración de procesos, *checkpoint-restart* (congelar uno o varios procesos, mudarlos de servidor y continuar su funcionamiento en el nuevo host), balanceo de carga, tolerancia a fallos, etc.;
✓ escalabilidad: debe poder detectar automáticamente nuevos servidores conectados al clúster para proceder a su utilización, o bien permitir añadirlos de forma manual.

Existen diversos tipos de middleware, como por ejemplo: MOSIX, OpenMOSIX, Condor, OpenSSI, MatLab, etc.

El middleware recibe los trabajos entrantes al clúster los redistribuye de manera que el proceso se ejecute más rápido y el sistema no sufra sobrecargas en un servidor. Esto se realiza mediante políticas definidas en el sistema (automáticamente o por un administrador) que le indican dónde y cómo debe distribuir los procesos, por un sistema de monitorización, el cual controla la carga de cada CPU y la cantidad de procesos en él.

El middleware también debe poder **migrar** procesos entre servidores con distintas finalidades:

• balancear la carga: si un nodo está muy cargado de procesos y otro está ocioso, pueden transferirse procesos a este último para liberar de carga al primero y optimizar el funcionamiento;
• mantenimiento de nodos: si hay procesos corriendo en un nodo que necesita mantenimiento o una actualización, es posible migrar los procesos a otro nodo y proceder a desconectar del clúster al primero;

- priorización de trabajos: en caso de tener varios procesos corriendo en el clúster pero uno de ellos de mayor importancia que los demás, puede migrarse este proceso a los nodos que posean más o mejores recursos para acelerar su procesamiento.

## 3.5 MATLAB DISTRIBUTED COMPUTING TOOLBOX*

**3.5.1 ¿Qué son los productos de cálculo distribuido?** La Toolbox de cálculo distribuido y el Motor de cálculos distribuidos de MATLAB (MATLAB Distributed Computing Engine, MDCE) permiten coordinar y ejecutar operaciones de MATLAB independientes de manera simultánea en un clúster de computadoras, acelerando la ejecución de trabajos grandes en MATLAB.

Un trabajo es alguna operación larga a realizar en MATLAB. Un trabajo se rompe en segmentos llamados tareas, esta división la decide el usuario. Bien podría dividirse el trabajo en tareas idénticas, pero éstas no tienen que serlo.

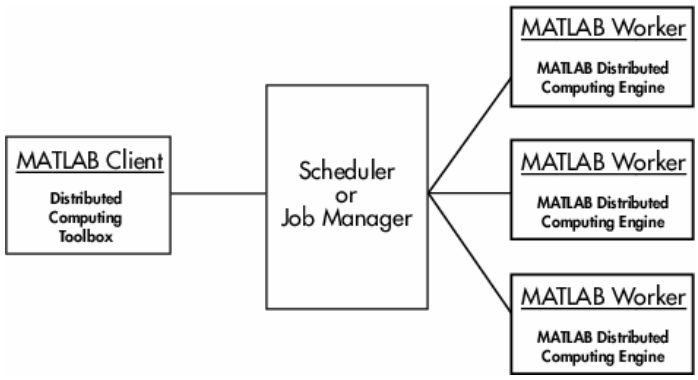
La sesión de MATLAB en la cual se define el trabajo y sus tareas es llamada la sesión cliente. Normalmente ésta ocurre en la máquina en donde el usuario programa con MATLAB. El cliente usa la Toolbox de cálculo distribuido para realizar la definición de los trabajos y las tareas. El motor de cálculo distribuido de MATLAB es el producto que realiza la ejecución del trabajo evaluando cada una de las tareas y entregando el resultado a la sesión cliente.

El Administrador de Trabajo o Job Manager es la parte del motor que coordina la ejecución de los trabajos y la evaluación de sus tareas. El Job Manager distribuye las tareas para su evaluación en cada una de las sesiones de MATLAB individuales del motor, llamadas trabajadores (workers). Usar el Job Manager de MathWorks (incluido en la Toolbox) es opcional; la distribución de tareas de los trabajadores puede realizarse por un despachador externo♦, tal como Windows CCS (Compute Cluster Server) o Platform LSF (Load Share Facility) que son plataformas distintas al MATLAB las cuales pueden correr el clúster.

* tomado de la toolbox de MATLAB

♦ fuente: http://download.microsoft.com/download/3/a/3/3a3b5bcd-e2c3-48da-ab77-9659bf1bc49a/Platform%20and%20CCS.pdf

Figura 4. Diagrama general del clúster




Fuente: Ayuda de Mathworks MATLAB 7.4

*Para determinar si está instalada en su sistema la Toolbox de cálculo distribuido, el usuario puede digitar este comando en la consola de MATLAB.*

```
ver
```

*Al ingresar este comando, MATLAB muestra información acerca de la versión de MATLAB que está ejecutando, incluyendo una lista de todas las toolboxes instaladas en el sistema y sus números de versión.*

*También puede ejecutarse el comando* `ver` *como parte de una tarea en una aplicación distribuida para determinar cuál versión del Motor de cálculo distribuido MATLAB (MDCE) está instalada en una máquina trabajadora. La toolbox y el MDCE deben tener la misma versión (compatibilidad).*

**3.5.2 Componentes de la toolbox y el motor.** *Los componentes de la toolbox y del motor de cálculo distribuido son los siguientes:*

- **Job Managers, Workers y Clientes**

*El administrador de trabajo o Job Manager puede correr en cualquier máquina en la red. El Job Manager ejecuta los trabajos en el orden en que sean indicados, a menos que alguno de los trabajos en cola sea cancelado, es decir, que uno de los workers no haya resuelto la tarea asignada.*

*A cada trabajador o* `worker`, *el Job Manager le asigna una tarea del trabajo en ejecución. Una vez ejecutada la tarea, el worker devuelve el resultado al Job*

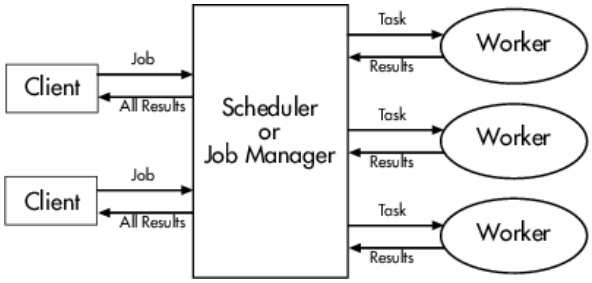
Manager, y entonces recibe otra tarea. Un worker va asociado a un único Job Manager. Cuando todas las tareas para un trabajo en ejecución han sido asignadas a los workers, el Job Manager comienza a ejecutar el siguiente trabajo con el siguiente worker disponible.

Normalmente una instalación del Motor de cálculo distribuido en MATLAB incluye muchos workers que pueden ejecutar todas las tareas simultáneamente, acelerando la ejecución de grandes trabajos en MATLAB. Generalmente no importa el worker que ejecuta una tarea específica. Los workers evalúan las tareas una a la vez, devolviendo los resultados al Job Manager. El Job Manager devuelve entonces los resultados de todas las tareas del trabajo en la sesión cliente.

Para probar las aplicaciones localmente o con otros propósitos, se puede configurar una única computadora como cliente, Job Manager y Worker. Además, puede tener más de una sesión worker o más de una sesión Job Manager en una misma máquina.

### Interacción de las sesiones de cálculo distribuido

Figura 5. Interacciones de cálculo distribuido con más de una sesión (cliente)

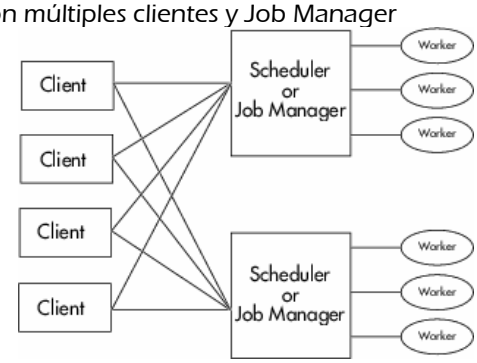



Fuente: Ayuda de Mathworks MATLAB 7.4

Una red grande puede contener varios Job Managers y varias sesiones cliente, el número de Job Managers es directamente proporcional al número de host que tenga la red. En la documentación del toolbox de MATLAB no se especifica un limite, este viene dado por el tiempo de procesamiento (overhead) intrínseco de la red y las necesidades del cliente. Cualquier sesión cliente puede crear, correr y acceder trabajos en cualquier Job Managers, pero una sesión worker está registrada y dedicada a sólo un Job Manager a la vez. La figura 6 muestra una configuración con múltiples Job Managers.

### Configuración Con Múltiples Clientes Y Job Managers

Figura 6. Configuraciones con múltiples clientes y Job Manager




Fuente: Ayuda de Mathworks MATLAB 7.4

*Una red grande puede incluir varios administradores de trabajo y varias sesiones cliente. Cualquier sesión cliente puede crear, correr y acceder trabajos en cualquier administrador.*

**3.5.3 Despachadores Externos (Schedulers).** *Como alternativa al Job Manager de MathWorks, puede usarse un despachador externo. Este puede ser*[*]*: Windows CCS, Platform Computing LSF, mpiexec, o un scheduler genérico.*

- **Eligiendo entre un Scheduler externo y el Job Manager.** *Al decidirse entre usar un Scheduler externo o el Job Manager de MathWorks, debería considerar lo siguiente:*

✓ **¿El clúster ya tiene un Scheduler?** *Si ya tiene un Scheduler, tal vez necesite usarlo como medio de control de acceso al clúster. El despachador existente puede ser tan fácil de usar como el Job Manager de MathWorks, así que puede no haber necesidad de involucrar administración extra.*

✓ **¿La única administración que se necesita es el manejo de trabajos de cálculo distribuido?** *El* **Job Manager** *de MathWorks está* **diseñado** *específicamente para aplicaciones de cálculo distribuido. Si no se necesitan otras tareas de programación, tal vez un despachador externo no ofrezca ninguna ventaja.*

✓ **¿El clúster ya está configurado para compartir archivos?** *El Job Manager de MathWorks puede administrar los recursos comunes e información necesaria para las*

[*] fuente: http://download.microsoft.com/download/3/a/3/3a3b5bcd-e2c3-48da-ab77-9659bf1bc49a/Platform%20and%20CCS.pdf

aplicaciones de cálculo distribuido. Esto podría ser útil en configuraciones donde el acceso compartido es limitado.

✓ **¿Se empleara el procesamiento administrado interactivo o en batch mode (grupos de transacciones a la vez)?** Cuando se usa Job Manager, los workers se mantienen en ejecución todo el tiempo, dedicando su trabajo Job Manager. Con un Scheduler externo, los workers corren como aplicaciones que comenzaron a evaluar las tareas, y se detienen cuando sus tareas se han completado. Si las tareas a ejecutar son pequeñas o consumen poco tiempo, iniciar un worker para cada máquina podría involucrar demasiado tiempo de *overhead (tiempo de procesamiento)*.

✓ **¿Hay precauciones con la seguridad?** Tal vez un Scheduler externo pueda configurarse más al acomodo de tales requerimientos particulares de seguridad.

✓ **¿Se necesita monitorear el progreso del trabajo o acceder datos intermedios?** La ejecución de un trabajo mediante el Job Manager soporta *eventos* y *callbacks*, así que pueden ejecutarse funciones particulares mientras cada trabajo y tarea progresa de un estado a otro.

• **Componentes en plataformas mixtas o Clústeres heterogéneos.** La Toolbox de cálculo distribuido y el motor de cálculo distribuido son soportados bajo las plataformas Windows, UNIX y Macintosh. Las plataformas mixtas estás soportadas, por tanto los clientes, administradores de trabajo y los trabajadores no tienen que estar montados sobre la misma plataforma. El clúster también puede implementarse en máquinas de 32-bit y 64-bit, mientras la información no exceda las limitantes impuestas por los sistemas de 32 bits.

En el entorno de una plataforma mixta, los administradores de sistema deben asegurarse de seguir las instrucciones de instalación adecuadas para la máquina local en la cual están instalando el software.

**3.5.4 Servicio del Motor de Cálculo Distribuido de MATLAB, MDCE.** Si está usando el Job Manager de MathWorks, cada máquina que aloje una sesión trabajador o de administrador de trabajo, también debe ejecutar el servicio del motor de cálculo distribuido de MATLAB (MDCE).

El servicio MDCE controla las sesiones worker y Job Manager, y las restaura cuando las máquinas anfitrión se bloquean. Cuando una máquina worker o una Job Manager se bloquea y reinicia, el servicio MDCE se reinicia (usualmente está configurado para iniciar durante la carga del sistema operativo) y restaura la sesión correspondiente justo antes

del colapso. Estos procesos son explicados profundamente en la Guía de Administradores de MDCE de la ayuda de MATLAB que se encuentra al abrir el software (MATLAB) y al oprimir los siguientes botones:

*Start\Toolboxes\Distributed Computing\Help*

**3.5.5. Componentes Representados en el Cliente.** Una sesión cliente se comunica con el Job Manager mediante la ejecución de métodos y configurando propiedades de un objeto de tipo "Job Manager". Aunque no es comúnmente necesario, la sesión cliente también puede acceder a información acerca de una sesión worker a través de un objeto de tipo "worker".

Cuando se crea un trabajo en la sesión cliente, el trabajo reside en el Job Manager o en el alojamiento de datos del Scheduler. La sesión de cliente tiene acceso al trabajo a través de un objeto tipo trabajo o "Job". Similarmente, las tareas que se definan para un trabajo en la sesión cliente existen en el Job Manager o en el alojamiento de datos del Scheduler, y puede acceder a ellos a través de objetos tipo tarea o "task".

## 3.6 MORFOLOGÍA DEL CLÚSTER EN MATLAB

El clúster bajo plataforma en Matlab está formado por los siguientes elementos:

- **Cliente MATLAB**

Es la sesión MATLAB que define y reparte el trabajo. Usualmente, esta es la sesión de MATLAB en la cual el programador desarrolla y modela las aplicaciones. También se conoce como el cliente MATLAB.

- **Computadora Cliente**

Es la computadora que ejecuta el cliente MATLAB.

- **Computadora (Ordenador)**

Un sistema con uno o más procesadores.

- **Labs**

Por defecto, cuando inician los workers, trabajan de forma independiente. Entonces pueden conectarse a los demás y trabajar como grupo, y se les conoce como *labs*.

- **MDCE**

Es el servicio que debe estarse ejecutando en **todas** las máquinas antes de que puedan correr como Job Manager o Worker. Este es el proceso origen del motor, asegurándose

de que los procesos de administrador de trabajo y trabajador siempre estén ejecutándose.

- **Nodo**

Una computadora que hace parte de un clúster.

- **Nodo Cabeza**

Usualmente es el nodo del clúster designado para ejecutar el Job Manager y el administrador de licencia. Frecuentemente, es útil correr todos los procesos no relacionados directamente con el trabajo en una sola máquina.

- ***Despachador –Scheduler-***

Es el proceso que organiza trabajos y asigna tareas a los trabajadores, ya sea de una empresa externa o de MathWorks, en cuyo caso es llamado JOB MANAGER.

- **Tarea (Task)**

Es un segmento de un trabajo, a ser evaluado por un worker.

- **Trabajo (Job)**

Es la operación completa a gran escala, compuesta de un conjunto de tareas, a realizar en MATLAB.

- **Trabajador (Worker)**

Es el proceso de MATLAB que realiza los cálculos de la tarea. También es Conocido como el trabajador MATLAB, proceso trabajador o simplemente "worker".

## *3.7 LA PLATAFORMA GRAFICA DEL MATLAB – GUI*

MATLAB admite la creación de interfaces gráficas de usuario (GUI, *Graphical User Interfaces*) le permitan a este último desentenderse de comandos y de sintaxis, automatizando y facilitando el uso final de una aplicación determinada. Esto se consigue mediante un conjunto de controles u objetos con diferentes características como botones, menús, ventanas, etc., que permiten manipular de manera muy simple programas efectuados dentro de este contexto. La programación orientada a objetos en MATLAB no está tan avanzada como en otros lenguajes de programación (LabVIEW, Visual Basic, Borland C, etc).

La preparación de GUIs se lleva a cabo en dos etapas: la primera de ellas consiste en manejar la herramienta de diseño de GUIs, contenida en el Matlab, llamada GUIDE; la

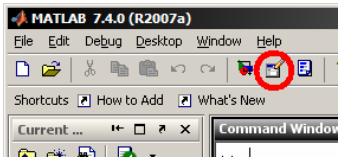
segunda parte consiste en programar los eventos (llamados callbacks) correspondientes a los controles y a la ventana en general.

Para lograr desarrollar programas que utilicen las capacidades gráficas de MATLAB hay que estar al tanto de algunos conceptos que se manifiestan en las siguientes secciones.

El cuadro GUI se crea en una ventana, identificada como figura y está constituida por los siguientes elementos:

- Menú de interfaz con el usuario
- Dispositivos de control de la interfaz con el usuario
- Ejes para desplegar las gráficas o imágenes.

Figura 7. Boton GUI



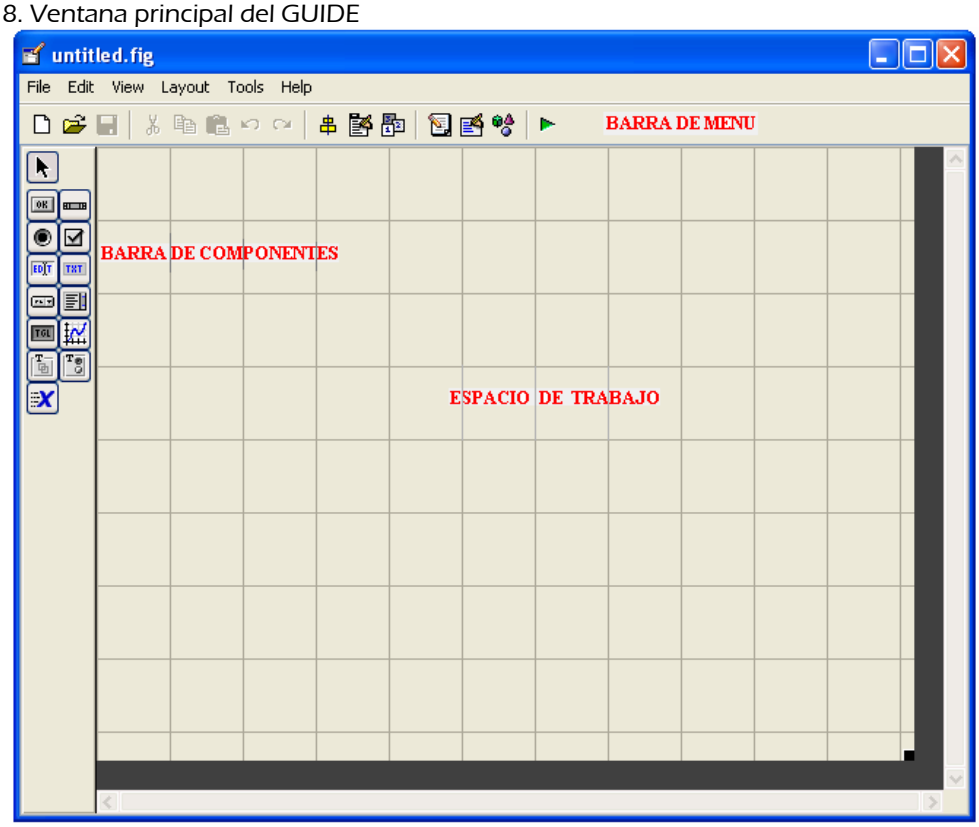

Por razón de la GUI, el flujo de información está controlado por las funciones (eventos) que sucedan en la interfaz. Comparando con los scripts, en estos los comandos están en un orden preestablecido, mientras que en la GUI no lo residen. Los comandos para crear una GUI se escriben en un script, pero una vez que se ejecuta la GUI, esta permanece en la pantalla si bien se haya terminado la ejecución del script. La interacción con el usuario se prolonga hasta que se cierra la GUI.

**3.7.1 Objetos de un GUI hecho en MATLAB.** El objeto más importante es la **ventana**, sobre el cual se ubican los demás. Es único en cada GUI. Una ventana puede tener también **controles** (uicontrols) tales como botones, barras de desplazamiento, botones de selección o de opción, etc.) y menús (uimenus). Adicionalmente, puede contener ejes (axes) que a su vez contendrán líneas y demás trazos. La jerarquía de objetos indica que en MATLAB hay objetos padres e hijos. Por ejemplo, cada ventana es padre de los objetos ejes, controles o menús que están por debajo.

**3.7.2 Utilizando GUIDE.** A la herramienta GUIDE se accede de varias maneras: tecleando el comando `guide` en la ventana de comando; usando el botón dispuesto para tal labor en la barra de herramientas (ver figura 7) o mediante el Menú `File>>New>>Gui`.

La ventana principal de GUIDE es la siguiente:

Figura 8. Ventana principal del GUIDE




Las Componentes principales de GUIDE son:

**Barra de Menús**: Aquí se encuentran las funciones elementales de Edición de GUI's.

**Paleta de Componentes** (component Palette): Aquí se encuentran los **uicontrols** estos componentes permite seleccionar los controles (objetos).

**Barra de Herramienta**: En ella se encuentran lo siguientes botones:

- Botón de ejecución (Run button): Al presionarse se crea la figura de la interfaz diseñada en el Layout Área.

- Alineación de Componentes (Alignment tool): esta opción permite alinear los componentes que se encuentran en el área de trabajo (Layout Área) de manera personalizada.

 Propiedades del Inspector (Property Inspector): con esta opción se asignan y modifican las propiedades de cada objeto en forma personalizada.

 Navegador de Objetos (Object Browser): Muestra todos los objetos que se encuentra en la figura (en forma de árbol) y a través de Object Browser se puede seleccionar los objetos.

 Editor de Menús (Menú Editor): El editor de Menú establece menús de ventana y menús de contexto.

Básicamente solo se necesita entender cinco comandos para poder describir una GUI: uimenu, uicontrol, get, set y axes. No obstante, lo que hace relativamente complicadas a estos comandos es el gran número de formas de uso que tiene. Es imposible describir todos lo tipo de situaciones, pues requiere demasiado espacio. Toda la información necesaria se encuentra en la documentación de MATLAB y puede verse bajo la sección **Building GUI** de la ayuda de MATLAB.

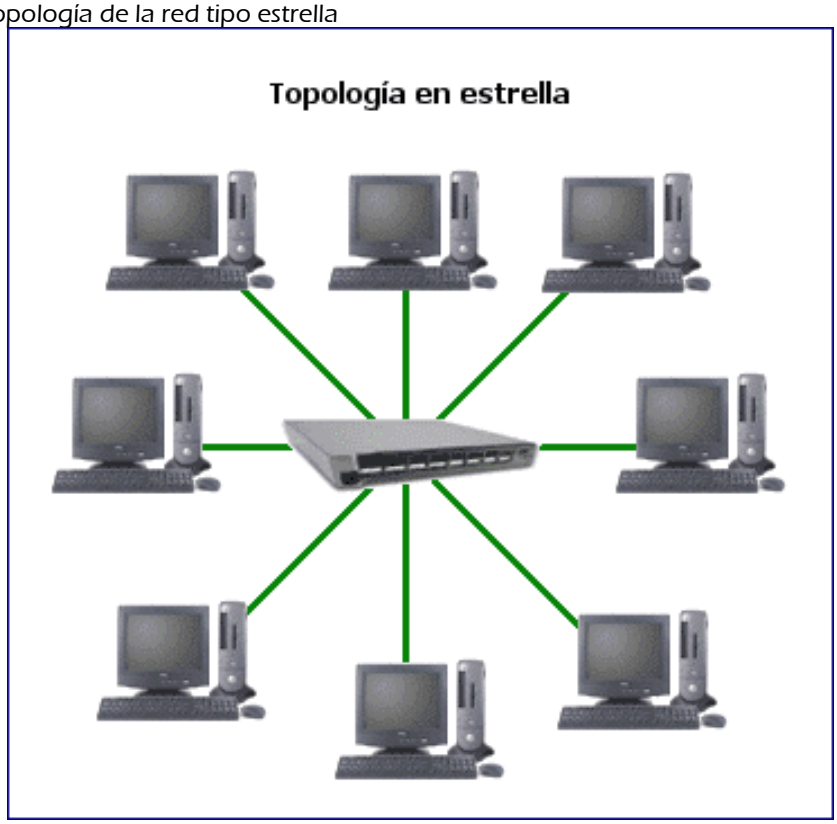
# 4. CARACTERÍSTICAS DE DISEÑO

Se segmentan las características del clúster en las siguientes categorías:

## 4.1 ESTRUCTURA FÍSICA.

El clúster de cálculo distribuido para el programa de Ingeniería Electrónica está conformado por 10 nodos ubicados en la sala de simulación del programa de ingeniería electrónica. La conexión entre los nodos del clúster se realiza mediante una red ethernet en topología estrella (ver figura 9) de 100Mbps, que a su vez es una subred de la red de la universidad llamada Simuele. El tráfico es controlado por un router central ubicado en la misma sala, que a su vez sirve de puerta de enlace cuando en los computadores de la sala se necesita acceso a la intranet de la universidad y a la Internet. Cada computador está conectado al router mediante su tarjeta de interfaz de red (NIC, Network Card Interface).

Figura 9. Topología de la red tipo estrella




Fuente: http://mx.geocities.com/alfonsoaraujocardenas/topologias.html

## 4.2 SISTEMA DE COMUNICACIONES.

Las comunicaciones entre nodos del clúster son manejadas por el Motor de cálculo distribuido de MATLAB (MDCE), quien necesita una cantidad de puertos disponibles para tal operación, de 5 puertos base por nodo + 1 puerto por cada nodo adicional en el clúster. Se requiere de al menos 5 puertos libres adicionales para comunicación inter-nodos. Adicionalmente, se requiere de un servicio de DNS que traduzca los nombres de los nodos en sus direcciones IP de red. Mathworks recomienda deshabilitar *firewalls* entre los nodos del clúster para asegurar una mejor respuesta de la red en general, disminuir las demoras de *overhead* y garantizar una comunicación más eficiente entre los nodos del clúster.

## 4.3 SOFTWARE DE MANEJO.

Todo el funcionamiento del clúster se basa en MATLAB; quien a su vez está instalado sobre el sistema operativo WINDOWS XP, quien resuelve los nombres de los hosts y administra la red privada en la que están conectados. Para la instalación, desinstalación y administración del clúster se desarrolló un software con GUI en MATLAB que simplifica considerablemente estas labores.

# 5. DESARROLLO DEL PROYECTO

## 5.1 DESCRIPCIÓN DEL FUNCIONAMIENTO DEL SISTEMA

El proyecto consiste en implementar un clúster de cálculo distribuido realizado bajo plataforma en MATLAB y con una interfaz gráfica (software) utilizando el GUI de MATLAB que es capaz dentro de una subred predeterminada realizar instalación, desinstalación y administración de un clúster. Este sistema es controlado desde el software de manejo y a través de un computador cliente quien se encarga de entregar y recibir las tareas asignadas a cada nodo del clúster ya preestablecido.

Al momento de arrancar el software, aparece el menú principal del programa con las siguientes 4 opciones:

- Instalar, desinstalar o reiniciar servicio.
- Instalar clúster.
- Desinstalar clúster.
- Consola de comandos.

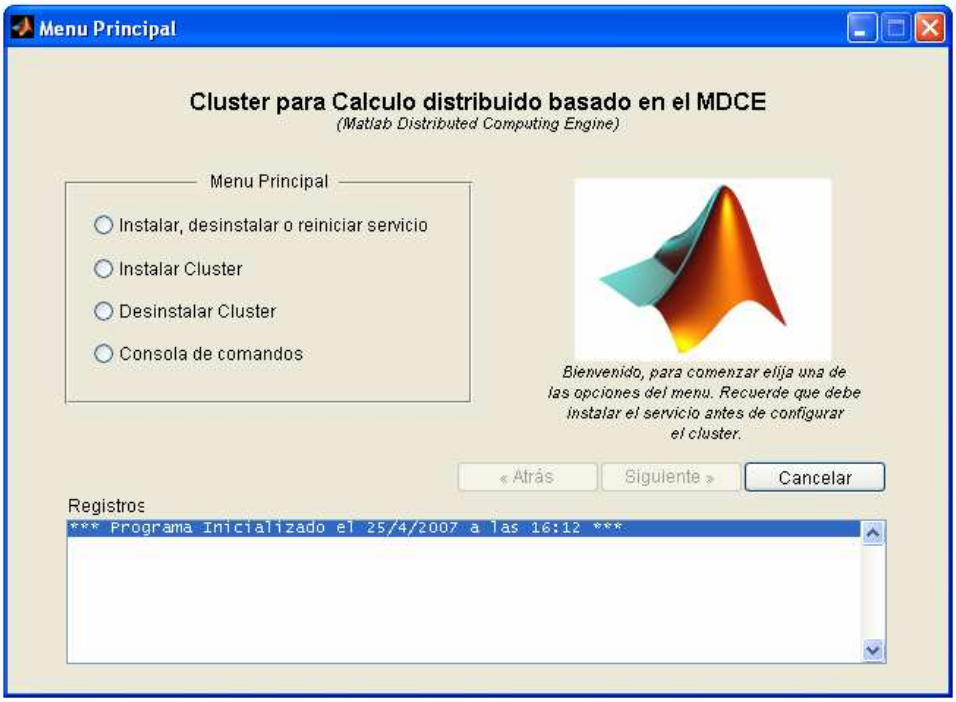
En la primera opción es donde se realiza la elección de arrancar el servicio MDCE de MATLAB el cual se encarga de comunicar los nodos entre si con el Job Manager, la segunda opción se refiere a la instalación del clúster según a la configuración que se le quiera dar, la siguiente se describe como desinstalación que se le dió al clúster ya predefinido anteriormente y por último una consola de comando para observar y reconfigurar detalles del clúster.

## 5.2 DISEÑO DEL PROGRAMA

El software de aplicación se realizó a través del programa MATLAB y específicamente recurrir a la utilidad del medio gráfico de dicho software llamado GUI. Durante su desarrollo además de instalar y configurar el clúster y afines, se enfatizó en poder ofrecer al usuario una forma sencilla, cómoda y agradable en el manejo del programa. Los archivos que le conforman deben ser ubicados dentro de la carpeta %MATLABROOT%\toolbox\distcomp\cluster. El script inicial es `scr00.m`, pero para hacer fácil su uso se recomienda usar el script `clustertesis.m`; el cual deberá ser ubicado dentro de la carpeta %MATLABROOT%\work o agregarlo al path del MATLAB en el cual se encuentra al abrir el software y dirigirse en File\Set Path\Add Folder.

Al ejecutarse el programa, se despliega la siguiente pantalla:

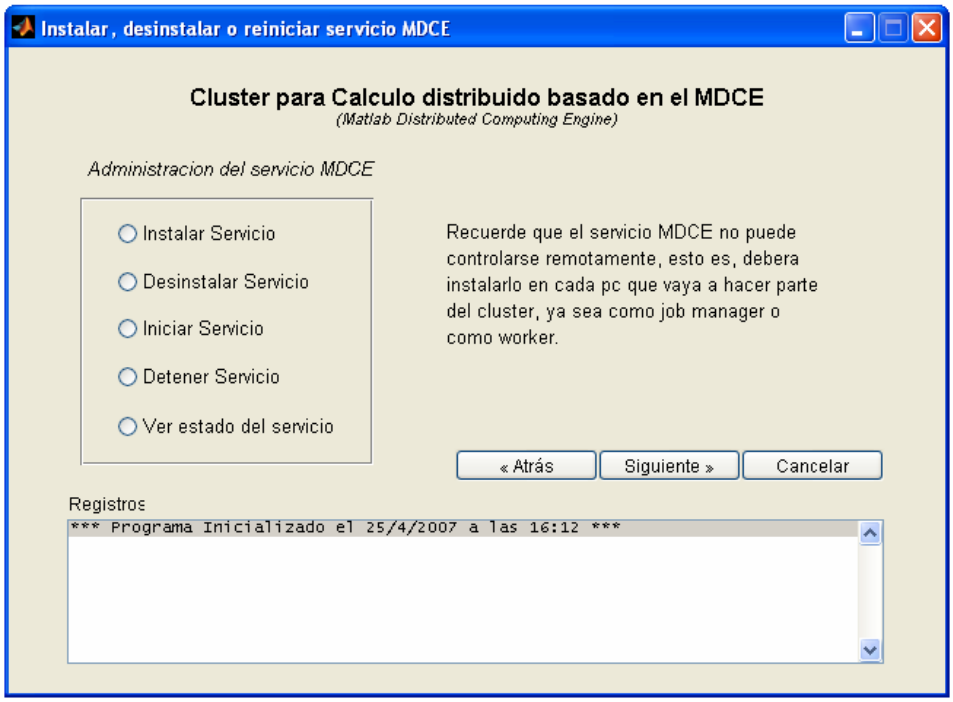
Figura 10. Menú principal




Como se observa, esta pantalla muestra el menú principal del programa con las cuatro (4) principales funciones, que permitirán al usuario administrar el servicio MDCE; además de instalar, desinstalar y hacer mantenimiento al clúster, desplegando comentarios e indicaciones según eventos que se presenten. Adicionalmente, durante todo el tiempo que dure ejecutándose la aplicación, se irá almacenando en un registro los eventos, comandos ejecutados y sus resultados.

## Módulo de administración del servicio MDCE

Figura 11. Módulo de administración del servicio MDCE

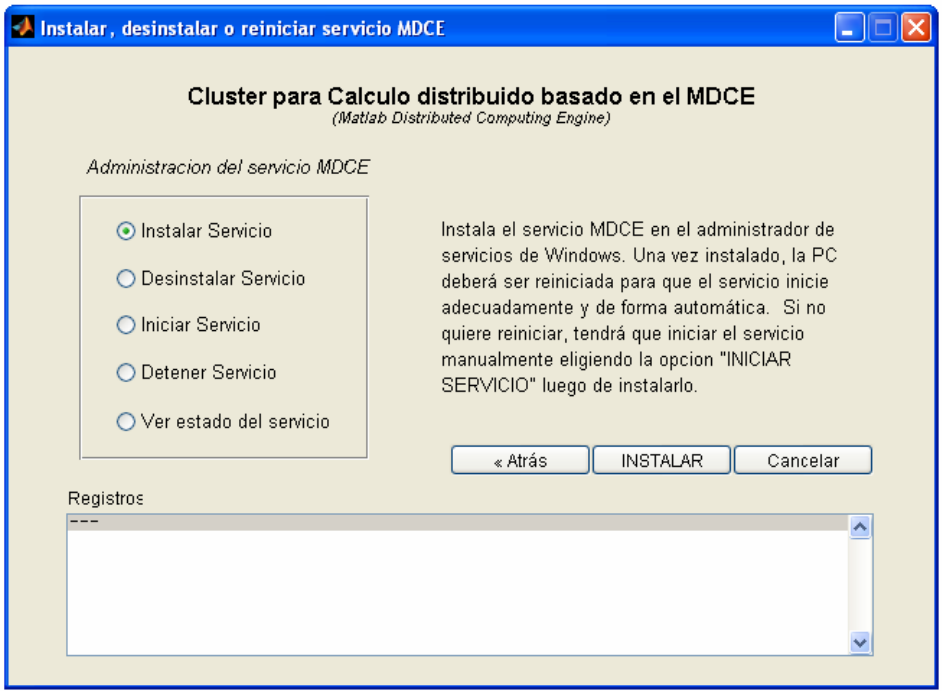



En este módulo se encuentran las funciones necesarias para la administración del servicio MDCE. Mediante estas funciones el usuario estará en la capacidad de instalar, desinstalar y controlar el servicio MDCE en el localhost (la pc actual en la que se ejecuta la aplicación). Además, puede monitorear el estado del servicio MDCE en cualquiera de los nodos del clúster; de esta forma se puede revisar que en TODOS los nodos esté instalado y corriendo el servicio, antes de proceder a instalar el clúster. Todas las actividades se van recopilando en el registro*.

Cabe recalcar que lo primero que debe hacer el usuario en este módulo es instalar el servicio MDCE previo a un inicio, detención o desinstalación del mismo, como lo muestra la figura 12:

* el registro se encuentra en la parte inferior del software como se muestra en la figura 11.

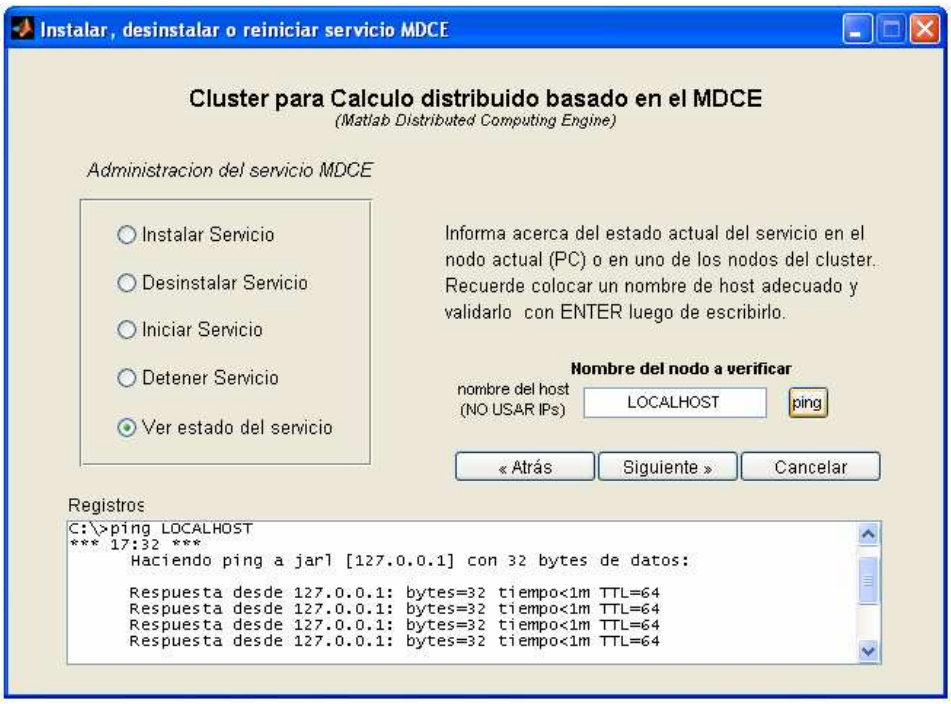
Figura 12. Módulo instalación de servicio




## Módulo ver estado de servicio

Este Módulo, como aparece en la siguiente figura 13, se presenta la opción **ver estado del servicio**, la cual permite al usuario visualizar el estado del servicio MDCE de cualquiera de los demás host de la subred que compone al clúster. Adicionalmente, se ofrece una opción para verificar la existencia de una conexión entre el host a revisar y la subred, mediante el comando **ping**. Tanto el clúster como el servicio de MDCE trabajan con nombres de hosts en lugar de IPs, esto garantiza la compatibilidad de los mismos en un entorno de IPs estáticas o dinámicas.

Figura 13. Módulo ver estado de servicio



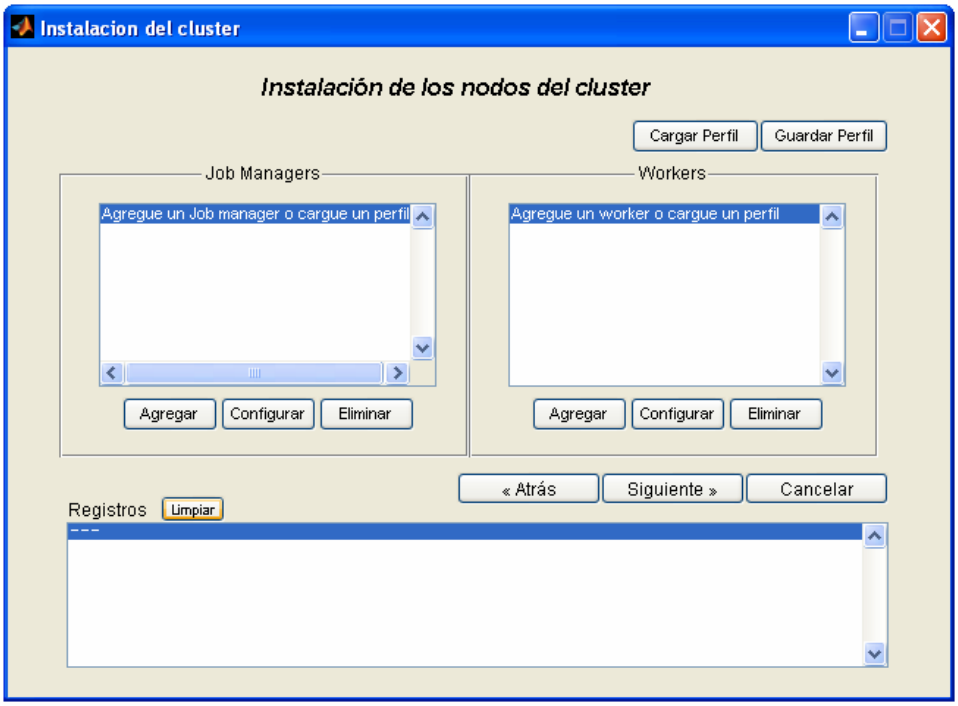

## Módulo instalación del clúster

Al indicar el botón de instalación del clúster del menú principal (ver figura 10), aparece el módulo de la figura 14. En este módulo se pueden agregar, configurar o eliminar los Job Managers o workers que conformarán el clúster de forma gráfica y sencilla.

Para facilitar la reinstalación de una configuración de clúster, se pueden guardar y cargar **perfiles** de configuración (Job Managers, Workers y sus hosts correspondientes) en archivos **.mat** (el formato de almacenamiento por defecto en MATLAB) que bien pueden ser almacenados en cualquier parte de la PC, o respaldados para su posterior reutilización.

Vale la pena anotar que para conseguir una desinstalación automática y sencilla del clúster es necesario haber almacenado el perfil de configuración del mismo, ya que de esta forma no será necesario desinstalarlo paso a paso usando la consola de comandos.

Figura 14. Módulo instalación del clúster




En la sección de Job Managers, al ejecutar la opción **agregar**, se habilita una ventana como se muestra en la figura 15. De manera muy similar se manifiesta la opción **configurar,** como aparece en la figura 16. En ambas ventanas podemos especificar o configurar según el caso los parámetros más relevantes de un Job Manager, el nombre y el host. Junto al parámetro host, se agregó un botón para hacer un ping de prueba y comprobar la existencia del host en la red y la conexión.

Con respecto a los workers, estos deben ser agregados y/o modificados solo cuando ha sido creado uno o más Job Managers, ya que TODO worker debe ir registrado con un y sólo un Job Manager correspondiente. Así que una vez se oprime la opción **agregar**, se abre una ventana como la de la figura 17 en la cual se deben introducir los parámetros del worker, además de especificar el Job Manager con el que el worker en creación irá registrado. De la misma forma se hace esto para la opción **configurar,** mostrada en la figura 18.

## Ventana de creación de un Job Manager

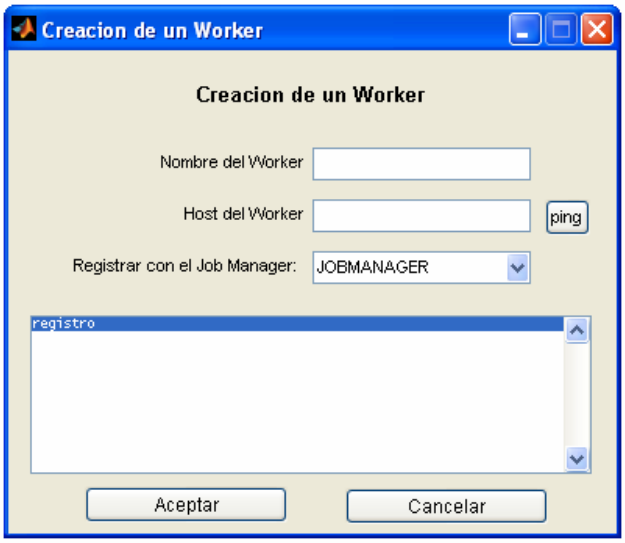
Figura 15. Ventana de creación de un Job Manager


## Ventana de configuración de un Job Manager

Figura 16. Ventana de configuración de un Job Manager

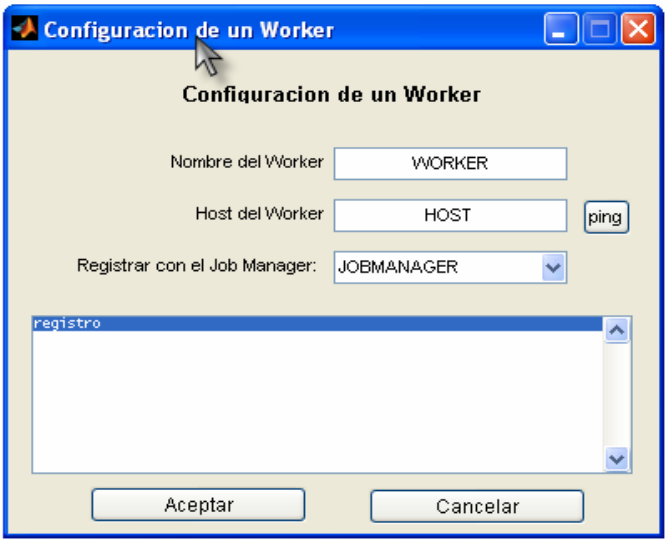
## Ventana de creación de un Worker

Figura 17. Ventana de creación de un Worker




## Ventana de configuración de un Worker

Figura 18. Ventana de configuración de un Worker



### Módulo desinstalar el clúster

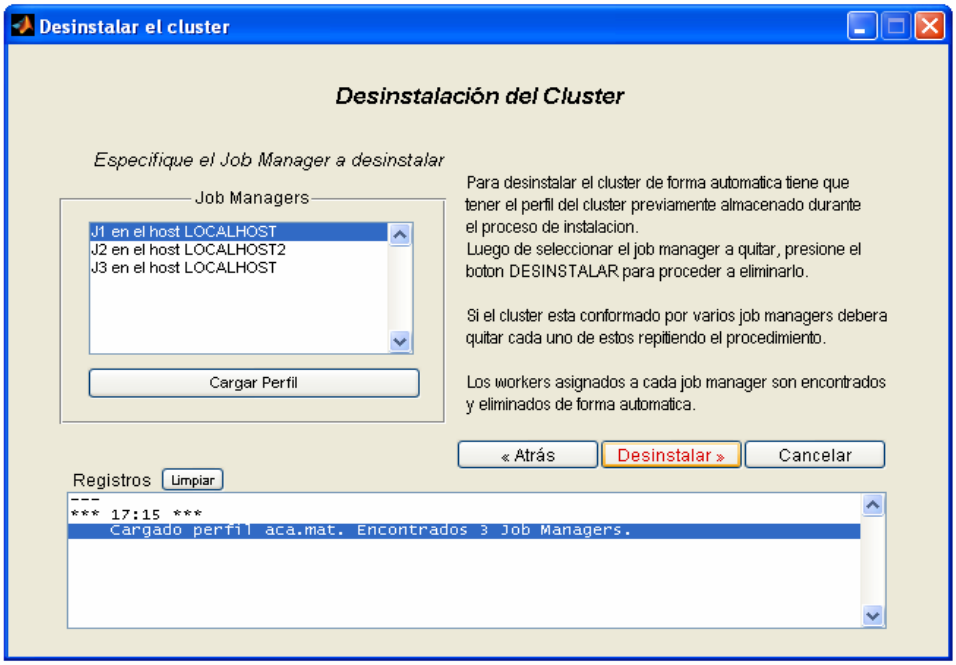
En esta sección, como se ve en la figura 19, el usuario puede desinstalar el clúster de forma automática a partir del perfil previamente almacenado durante el procedimiento de instalación. Si el clúster está conformado por varios Job Managers, estos deben ser desinstalados uno a uno. Los workers asociados a éste son encontrados y desinstalados.

Figura 19. Módulo desinstalar el clúster


En caso de errores irrecuperables como el bloqueo de los nodos del clúster, no será posible detener los nodos de forma automática, por lo que el administrador del clúster deberá hacerlo de forma manual mediante la consola de comandos asistida.

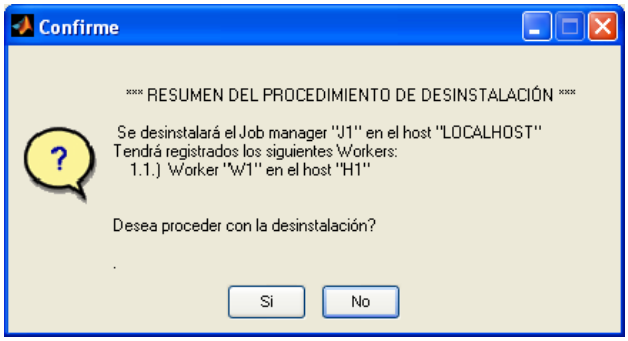
Para desinstalar el clúster con este módulo debe primero cargarse el perfil almacenado en el disco duro. Posteriormente, se elije el Job Manager a desinstalar, como muestra la figura 20:

Figura 20. Módulo desinstalar el clúster cuando se carga un perfil



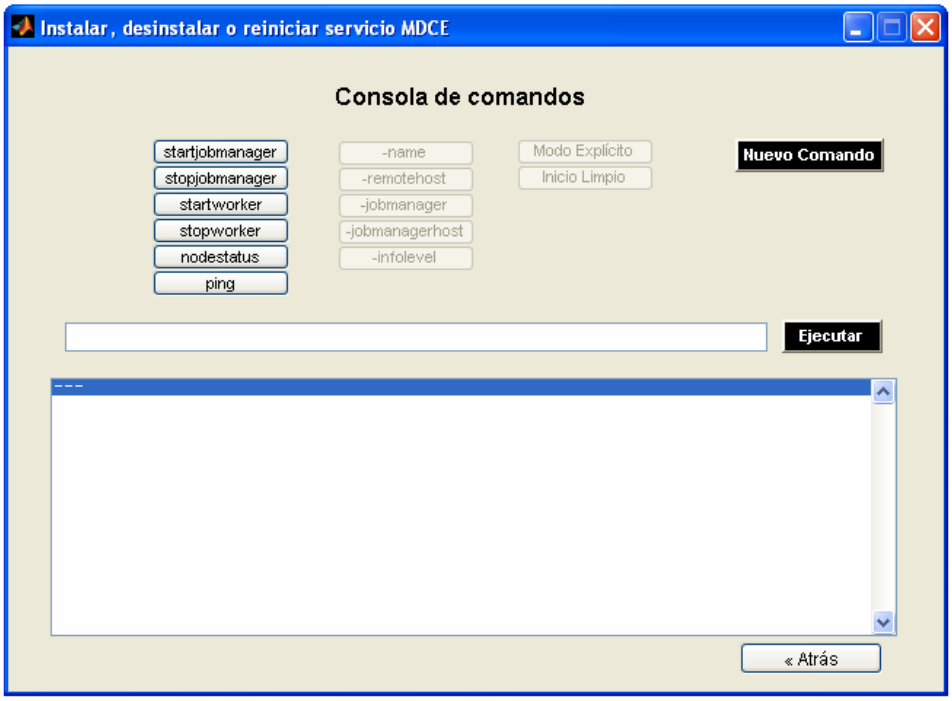

Finalmente, al darle un clic en la opción desinstalar, se muestra una ventana de confirmación de desinstalación como se muestra en la figura 21. Posteriormente se amplía el registro para mostrar todos los pasos ejecutados y su respectiva respuesta.

Figura 21. Ventana de confirmación de desinstalación



## Módulo de consola de comandos

Figura 22. Consola de comandos




La Toolbox de cálculo distribuido de MATLAB ofrece un conjunto de comandos de consola (archivos .bat, no scripts de MATLAB) para realizar las tareas de instalación y mantenimiento del clúster. Bajo condiciones de instalación y desinstalación normales, estos comandos son ejecutados de forma transparente al usuario, es decir, el administrador del clúster no tiene que conocerlos ni ver cómo se manejan (no obstante, siempre que se ejecute un comando tanto el resultado como el comando que lo provocó son almacenados en el registro).

Gracias a los botones disponibles que se observan en la figura 22, se minimiza la posibilidad de errores humanos de digitación y se facilita el uso de los comandos de forma que es posible:

Iniciar un Job Manager:

```
startjobmanager –name <NOMBRE> -remotehost <HOST> -v
```

Detener un Job Manager:

```
stopjobmanager –name <NOMBRE> -remotehost <HOST>  -v
```

Iniciar un trabajador o worker:

```
startworker –name <NOMBRE> -remotehost <HOST> -jobmanager ...
 <JOBMANAGER> -jobmanagerhost <HOSTDELJOBMANAGER> -v
```

Detener un trabajador o worker:

```
stopworker –name <NOMBRE> -remotehost <HOST> -v
```

Ver el estado de un nodo:

```
nodestatus –remotehost <HOST>  -infolevel <1,2,3>
```

Hacer ping a un host del clúster (probar conexión)

```
Ping <HOST>
```

La opción `-v` permite ver de forma explícita qué pasa durante la ejecución del comando.
La opción `-infolevel` # especificar qué tanta información quiere verse (1, 2 o 3).

Para obtener una documentación más extendida acerca de estos comandos de control, puede consultarse la ayuda de MATLAB que se encuentra al abrir el software (MATLAB) y al oprimir los siguientes botones:

*Start\Toolboxes\Distributed Computing\Help*

Y buscar cada comando en la opción: *Search for*

# 6. PRUEBAS DE RENDIMIENTO

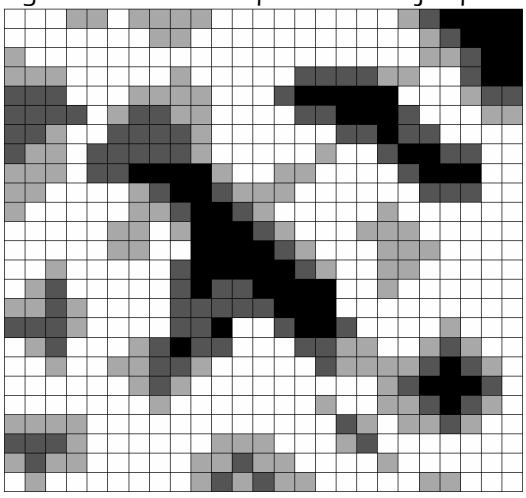
Para comprobar el funcionamiento y eficacia del clúster se decidió emplear un ejemplo computacionalmente pesado, con un tiempo de ejecución lo suficientemente largo como para poder demostrar los beneficios de segmentar el código, paralelizarlo en tareas y ejecutarlo en el clúster.

## 6.1 ALGORITMO EMPLEADO

El problema planteado es el siguiente: Se tiene un autómata el cual debe avanzar por un terreno segmentado en 25 x 25 secciones y cuyo grado de escarpado o dificultad se ha calificado entre 0, 10, 20 y 30 (siendo 30 el mayor grado de dificultad). Este grado de dificultad es proporcional al tiempo que requiere el autómata para avanzar y por tanto a la potencia empleada por el mismo. Este terreno debe recorrerse desde el sector de arriba a la izquierda al sector de abajo a la derecha, usando el camino más sencillo (menos escarpado) posible, minimizando el tiempo empleado y la potencia consumida para recorrerlo.

Se eligió emplear algoritmos genéticos para calcular posibles rutas sencillas, ya que son computacionalmente pesados (tienen un alto costo computacional) y son tareas fácilmente separables (segmentables), esto con el único objetivo de poder realizarle pruebas al clúster de cálculo distribuido y ver la real eficiencia que podría tener el sistema como tal.

Figura 23. Terreno empleado en el ejemplo

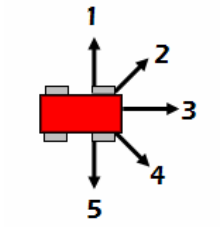


| | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 10 | 10 | 0 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 20 | 30 | 30 | 30 | 30 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 20 | 30 | 30 | 30 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 20 | 30 | 30 |
| 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 0 | 0 | 0 | 0 | 20 | 20 | 20 | 20 | 10 | 10 | 0 | 0 | 20 | 30 | 30 |
| 20 | 20 | 20 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 20 | 30 | 30 | 30 | 30 | 30 | 0 | 0 | 0 | 10 | 20 | 20 |
| 20 | 20 | 20 | 20 | 0 | 10 | 20 | 20 | 10 | 10 | 0 | 0 | 0 | 0 | 20 | 20 | 30 | 30 | 30 | 20 | 0 | 0 | 0 | 10 | 10 |
| 20 | 20 | 10 | 0 | 0 | 20 | 20 | 20 | 20 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 20 | 20 | 30 | 20 | 20 | 0 | 0 | 0 | 0 |
| 20 | 10 | 10 | 0 | 20 | 20 | 20 | 20 | 20 | 10 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 0 | 20 | 30 | 30 | 20 | 20 | 0 | 0 |
| 10 | 10 | 10 | 0 | 20 | 20 | 30 | 30 | 30 | 30 | 10 | 0 | 0 | 10 | 10 | 0 | 0 | 0 | 0 | 20 | 30 | 30 | 30 | 0 | 0 |
| 10 | 10 | 0 | 0 | 0 | 0 | 10 | 20 | 30 | 30 | 20 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 20 | 20 | 20 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 20 | 30 | 30 | 20 | 10 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 10 | 10 | 0 | 10 | 30 | 30 | 30 | 20 | 10 | 0 | 0 | 0 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 10 | 10 | 0 | 0 | 30 | 30 | 30 | 30 | 20 | 10 | 0 | 0 | 0 | 10 | 10 | 10 | 0 | 0 | 0 | 0 |
| 0 | 0 | 10 | 0 | 0 | 0 | 0 | 0 | 20 | 30 | 30 | 30 | 30 | 30 | 20 | 10 | 0 | 0 | 10 | 10 | 0 | 0 | 0 | 0 | 0 |
| 0 | 10 | 20 | 0 | 0 | 0 | 0 | 0 | 20 | 30 | 20 | 20 | 30 | 30 | 30 | 30 | 0 | 0 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 10 | 20 | 10 | 0 | 0 | 0 | 0 | 20 | 20 | 20 | 20 | 20 | 30 | 30 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | 20 | 20 | 10 | 0 | 0 | 0 | 0 | 20 | 20 | 30 | 0 | 0 | 20 | 30 | 30 | 20 | 0 | 0 | 0 | 0 | 10 | 0 | 0 | 0 |
| 0 | 10 | 20 | 0 | 0 | 0 | 10 | 20 | 30 | 20 | 20 | 0 | 0 | 0 | 20 | 20 | 10 | 10 | 0 | 0 | 10 | 20 | 10 | 0 | 0 |
| 0 | 0 | 10 | 0 | 0 | 10 | 10 | 20 | 20 | 10 | 0 | 0 | 0 | 0 | 0 | 20 | 10 | 10 | 10 | 10 | 20 | 30 | 20 | 10 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 10 | 20 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 20 | 30 | 30 | 30 | 20 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 0 | 10 | 10 | 20 | 30 | 20 | 10 | 0 | 0 |
| 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 20 | 10 | 0 | 10 | 10 | 20 | 10 | 0 | 0 | 0 |
| 20 | 20 | 20 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 0 | 0 | 0 | 10 | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 20 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 20 | 10 | 10 | 0 | 0 | 0 | 0 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 20 | 10 | 20 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 0 | 0 | 0 |

A modo de nomenclatura, se decidió asignar un número a cada una de las direcciones que pueda tomar el autómata. Por simplicidad de código y enfoque del problema, se

optó por usar sólo algunas de las 8 posibles direcciones (ver figura 24). Las rutas se representan como vectores fila de números de uno (1) a cinco (5).

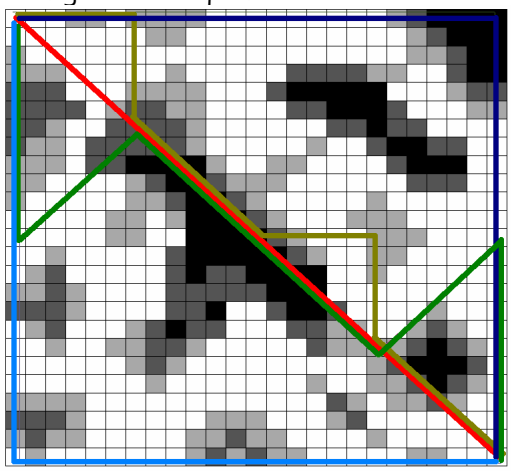
Figura 24. Nomenclatura de las direcciones del autómata




El objetivo, entonces, es minimizar el esfuerzo que debe hacer el autómata para completar el recorrido. Para determinarlo, se diseñó una función llamada *camino*, que recibe como parámetro de entrada la ruta completa a seguir (vector fila de direcciones del autómata) y entrega como valor de salida un valor entero proporcional a la dificultad vista encontrada en todo este recorrido por el terreno. Esta es la función a minimizar mediante algoritmos genéticos.

Para encontrar las rutas más sencillas se emplearán tres (3) algoritmos genéticos, configurados de forma diferente: distintas funciones de cruce, de mutación, y técnicas de migración. La población inicial es la misma: cinco (5) rutas arbitrarias que van de la esquina superior izquierda a la esquina inferior derecha, como lo muestra la figura 25.

Figura 25. Rutas iniciales de los algoritmos empleados



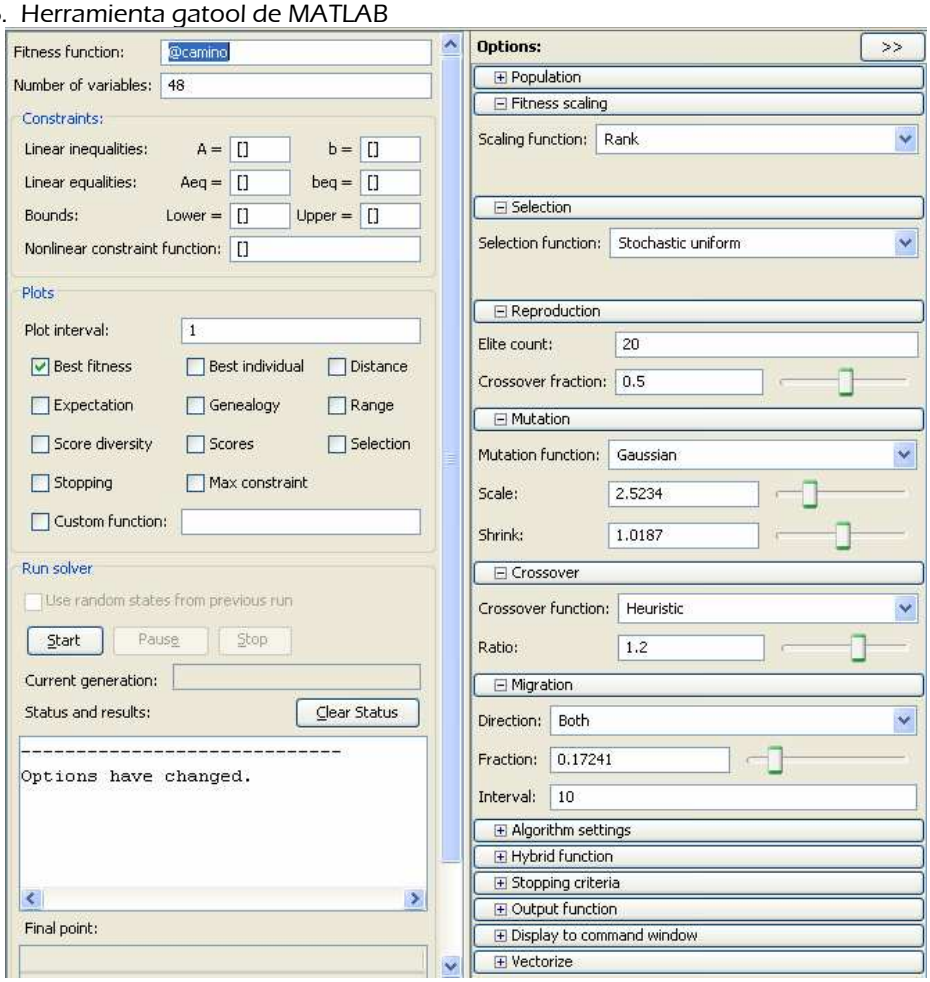
Para las rutas iniciales, el valor inicial de la función camino es el siguiente:

**Tabla 1. Valores de la función camino para la población inicial**

| Ruta | Valor inicial función Camino |
|---|---|
| 1: Azul Oscuro | 426 |
| 2: Amarillo Ocre | 464 |
| 3: Roja | 462 |
| 4: Verde | 610 |
| 5: Azul Claro | 406 |

Para la configuración de tres (3) los algoritmos genéticos a emplear se usó la herramienta `gatool` de MATLAB. Esta herramienta permite la importación y exportación de configuraciones detalladas de algoritmos genéticos

Figura 26. Herramienta gatool de MATLAB




Las tres configuraciones **difieren solamente en la función de cruce**: heurística, esparcida, e intermedia. Se eligen esas opciones para poder hacer una comparación entre los diferentes resultados obtenidos.

Luego de creadas las configuraciones (también conocidas como **problemas**), se exportaron como archivos **.mat** que serán cargados en el momento de la ejecución de las versiones paralela y no paralela del código

## 6.2 PARALELIZACIÓN DEL CÓDIGO

Para hacer que este código corra en el clúster es necesario analizarlo detenidamente. Si se separa el trabajo en muchas tareas sencillas, se corre el riesgo de hacer que el clúster gaste más tiempo transmitiendo y recibiendo datos entre los nodos que realizando los cómputos; así que la idea es conseguir un equilibrio entre la sencillez y la cantidad de tareas a realizar. El problema implica evaluar tres (3) algoritmos genéticos diferentes cada uno de ellos con cinco (5) poblaciones iniciales distintas durante quinientas (500) generaciones. Se optó por asignar este límite ya que encontrar una solución ideal podría ser algo que nunca llegase, y exigir un grado de minimización mayor puede hacer que el algoritmo tarde demasiado tiempo.
Se decidió por asignar una (1) tarea a cada uno de estos quince (15) procedimientos: la evaluación de cada uno de los algoritmos genéticos con cada una de las rutas o poblaciones iniciales. Esto se hizo no solo por ser procedimientos simétricos (evaluarlos toma aproximadamente la misma cantidad de tiempo) sino por ser razonablemente 'pesados' para ser resueltos por cada trabajador.

Para crear un trabajo y ejecutarlo en el clúster se debe operar a través de objetos. Así que seguimos el siguiente procedimiento:

- ✓ Encontrar un Job Manager disponible con la función `findResource.`
- ✓ Crear un objeto tipo Trabajo (Job) con la función `createJob.`
- ✓ Crear las tareas con la función `createTask.`
- ✓ Cargar el trabajo en la cola del Job Manager mediante la función `submit.`
- ✓ Esperar a que el trabajo termine su ejecución mediante la función `waitForState.`
- ✓ Recoger los resultados del trabajo mediante la función `getAllOutputArguments.`

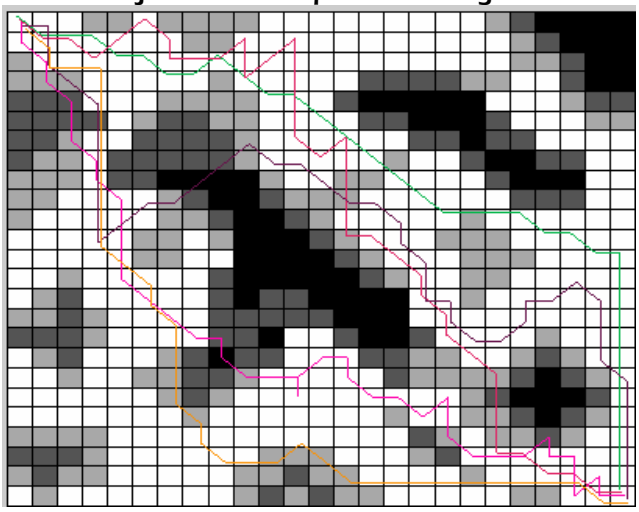
Toda la documentación sobre las diferentes formas de paralelizar código y de ejecutar trabajos en el clúster se encuentra disponible en la documentación de la Toolbox de cálculo distribuido, en la ayuda de MATLAB.

## 6.3 RESULTADOS OBTENIDOS

Los resultados obtenidos, tanto en la versión paralela como la versión no paralela del ejemplo, son los mismos: 15 rutas optimizadas, en otras palabras, los resultados son los mismos para las dos versiones.

El script genera tres (3) gráficas para las dos versiones (paralela y no paralela) de manera independiente como se observa en las figura 27, figura 28 y figura 29 (una para cada configuración de algoritmo genético (AG)), cada una con las 5 rutas ya modificadas por el paso de 500 generaciones*.

**Figura 27. Rutas finales tras la ejecución de la primera configuración de AG.**



* se optó por 500 el número de generaciones por el equilibrio presentado entre su alto coste computacional y el tiempo empleado en ejecutarse el algoritmo. Encontrar una solución ideal podría ser algo que nunca llegase dada la naturaleza aleatoria de los algoritmos genéticos, y exigir un grado de minimización mayor hacía que el algoritmo tardase demasiado tiempo en ejecutarse, lo cual resultaba poco práctico dado el volumen de pruebas que se debían efectuar.

**Figura 28. *Rutas finales tras la ejecución de la segunda configuración de AG.***

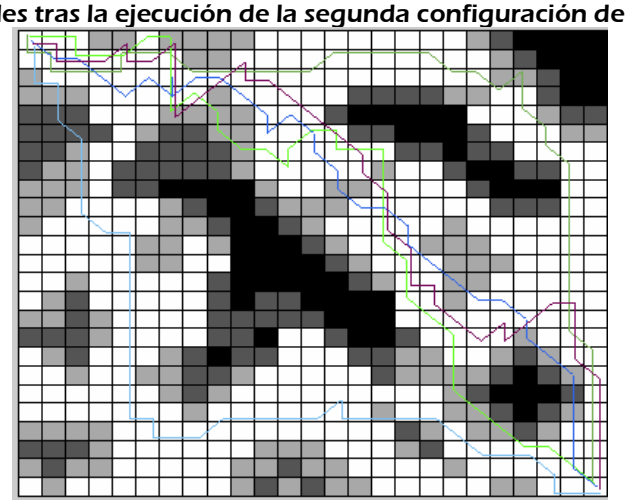
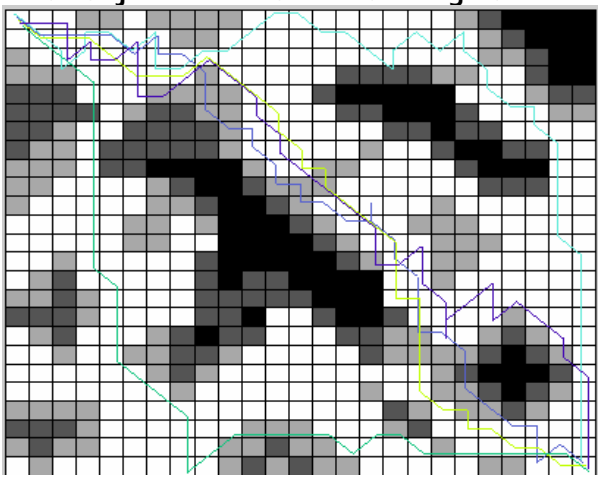


**Figura 29. *Rutas finales tras la ejecución de la tercera configuración de AG.***



Con respecto a los tiempos, los mismos scripts se encargaban de medir su propio tiempo de ejecución, mediante las funciones **`tic`** y **`toc`** de MATLAB: ejecutamos `tic` al iniciar la ejecución y `toc` al final de la misma. El tiempo empleado por la versión NO PARALELA (convencional) del ejemplo, fue ***256,1279 segundos*** (4'16.13").

| Tiempo Versión NO PARALELA (seg) | 256,13 |
|---|---|

Al ejecutar la versión PARALELA del ejemplo, se hicieron dos 'tomas' de tiempo y se promediaron (usando la media aritmética); repitiendo esta prueba para una cantidad de trabajadores/workers variada desde 1 hasta 10. Los resultados obtenidos fueron los siguientes:

**Tabla 2. Comparación de los resultados obtenidos con el clúster**

| workers | Tiempo_1(seg) | Tiempo_2(seg) | Promedio | Tiempo empleado* | Rapidez |
|---|---|---|---|---|---|
| 1 | 292,09" | 292,09" | 292,09" | 114,04 % | 0.87 |
| 2 | 137,65" | 137,65" | 137,65" | 53,74 % | 1.86 |
| 3 | 106,67" | 106,59" | 106,63" | 41,63 % | 2.40 |
| 4 | 95,56" | 96,39" | 95,97" | 37,47 % | 2.66 |
| 5 | 76,83" | 78,27" | 77,55" | 30,28 % | 3.30 |
| 6 | 71,04" | 70,38" | 70,71" | 27,61 % | 3.62 |
| 7 | 61,27" | 57,60" | 59,44" | 23,21 % | 4.30 |
| 8 | 54,67" | 56,02" | 55,35" | 21,61 % | 4.62 |
| 9 | 55,60" | 52,36" | 53,98" | 21,07 % | 4.74 |
| 10 | 63,94" | 64,94" | 64,44" | 25,16 % | 3.97 |

* Con respecto a la versión NO PARALELA del ejemplo

Se define el Tiempo empleado como el porcentaje de tiempo que necesitó el cluster para ejecutar el algoritmo paralelizado con respecto al tiempo de ejecución de la versión no paralela del mismo. Por ejemplo, para el caso de 2 workers se debe interpretar esta relación afirmando que el cluster ejecutó el algoritmo paralelizado en un poco más de la mitad del tiempo empleado por la versión secuencial.

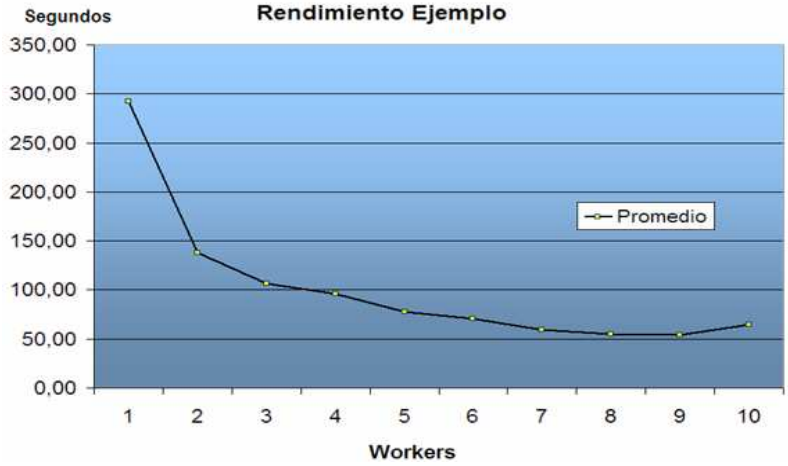
$$Tiempo\ empleado = \frac{promedio\ de\ las\ tomas}{Tiempo\ \ version\ \ no\ paralela} \times 100\%$$

Definimos la rapidez como la relación directa entre el tiempo de ejecución del algoritmo en modo secuencial y el tiempo de ejecución en el cluster. Nos dice qué tan rápida es la configuración del cluster con respecto a la velocidad de la versión no paralela. Así, una rapidez de 2 significaría que la configuración es dos veces más rápida que la versión no paralela.

$$Rapidez = \frac{Tiempo\ \ version\ \ no\ \ paralela}{promedio\ \ de\ las\ tomas}$$

Este parámetro muestra qué tan eficiente resulta el cluster en el uso del tiempo al ejecutar el algoritmo, al compararlo con la versión no paralela.

**Figura 30. Gráfica de rendimiento del clúster al ejecutar el ejemplo**




En la grafica tenemos en el eje de las abscisas (X) la cantidad de trabajadores empleadas, y en el eje de las ordenadas (Y) la cantidad de segundos. La serie mostrada en la grafica es el promedio de las tomas de tiempo de ejecución del algoritmo.

Una vez evaluados los tiempos, evaluaremos los resultados obtenidos: esto es, el valor final de la función a minimizar (camino) luego del paso de las 500 generaciones, tanto para la versión no paralela como para la versión ejecutada en distintas configuraciones del cluster; para las tres configuraciones de algoritmos genéticos (**AG**) empleadas.

### Tabla 3. Valores finales de la función a minimizar

| Ruta | VALOR INICIAL | VALORES FINALES FUNCIÓN CAMINO LUEGO DE 500 GENERACIONES | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Versión NO paralela | | | V. Paralela con 2 Workers | | | V. Paralela con 4 Workers | | | V. Paralela con 6 Workers | | |
| | | AG1 | AG2 | AG3 | AG1 | AG2 | AG3 | AG1 | AG2 | AG3 | AG1 | AG2 | AG3 |
| 1: Azul Oscuro | **426** | 103 | 142 | 131 | 96 | 110 | 144 | 104 | 176 | 200 | 154 | 294 | 135 |
| 2: Amarillo Ocre | **464** | 187 | 161 | 159 | 183 | 198 | 224 | 197 | 137 | 222 | 166 | 229 | 165 |
| 3: Roja | **462** | 177 | 269 | 167 | 251 | 172 | 136 | 163 | 166 | 194 | 128 | 177 | 182 |
| 4: Verde | **610** | 213 | 184 | 204 | 221 | 141 | 153 | 324 | 136 | 263 | 146 | 275 | 198 |
| 5: Azul Claro | **406** | 192 | 195 | 153 | 190 | 201 | 165 | 156 | 207 | 195 | 214 | 208 | 220 |

## 6.4 ANALISIS DE RESULTADOS

Para el ejemplo desarrollado, los resultados demuestran que a medida que se incrementa cada vez mas la cantidad de trabajadores en el clúster, las mejoras en el rendimiento no son notables (decrecen). Esto se debe a la comunicación adicional requerida (Overhead) al adicionar un nuevo trabajador, y también al grado de segmentación del problema entre los trabajadores (granulosidad). Un trabajo más granulado puede ejecutar mejor si se dispone de más trabajadores, pero para cantidades de nodos pequeñas puede resultar más eficiente el segmentar tareas más grandes.

Es normal que el trabajo corra más rápido en una PC de forma independiente que en un clúster con un solo trabajador, porque en este último caso se requiere la creación de las tareas y la comunicación de parámetros de entrada y de salida a través de la red, los cuales traen consigo un retardo.

Tener una cantidad de trabajadores alta, no garantiza que el clúster sea más eficiente, es la experiencia del programador a la hora de segmentar el código la que determina qué tan bien se aprovechen estos recursos.

Por otra parte, pese a la naturaleza aleatoria de los algoritmos genéticos, los resultados de la minimización de la función **camino** mantienen la misma tendencia y son de magnitudes similares tanto en la versión paralela como la no paralela; presentando un grado de minimización mayor en el algoritmo genético uno (1). Esto demuestra que el cluster no solo procesa la información más rápidamente sino que lo hace de forma correcta y los resultados obtenidos son igualmente confiables a los obtenibles mediante un algoritmo secuencial.

## 7. CONCLUSIONES

- Mediante la aplicación desarrollada en GUI no solo fue posible implementar el clúster de cálculo distribuido en la sala de simulación, sino que se creó una herramienta sencilla pero poderosa para administrarlo con mucha mayor facilidad a emplear comandos de consola de sistema (cmd).

- Siempre que se vaya a implementar el clúster, debe garantizarse que habrá libre comunicación entre los nodos que la conforman. Esto se logra desactivando los muros cortafuegos (firewalls) de forma que todos los nodos del clúster estén del mismo lado del muro.

- Al usar nombres de nodo es importante tener en cuenta que MATLAB no es compatible con todos los caracteres que el Windows posee, por lo que debe evitarse el uso de rayas bajas o *underscore* ("_"), guiones, espacios en blanco y demás. De ser posible, deben emplearse solo caracteres alfanuméricos.

- Los factores que garantizan que el clúster sea aprovechado y empleado de forma eficiente son la experiencia del programador y el conocimiento que este tenga acerca de la naturaleza del problema a resolver y la forma en que este se aborda.

- Si la aplicación no es muy compleja (sencilla), se recomienda optimizar la aplicación para que se ejecute de forma rápida en una sola PC y no necesitar paralelizarla (utilizar el cluster), ya que hacer esto añade retardos adicionales por transmisión de datos, comunicación entre los nodos y entre los trabajadores y el Job Manager.

- El cluster no solo procesa la información más rápidamente sino que lo hace de forma correcta, ya que los resultados obtenidos con él son igualmente acertados y confiables a los obtenibles mediante un algoritmo secuencial.

- Una mayor cantidad de trabajadores es una condición necesaria mas no suficiente para garantizar una mayor velocidad de procesamiento del clúster. Es la granulosidad de la aplicación (el grado de complejidad de cada tarea individual) la que debe ir acorde a la cantidad de trabajadores disponibles, a la complejidad y al tipo de problema que se pretende resolver mediante programación paralela en MATLAB.

- *MATLAB es una herramienta sumamente poderosa y útil para cualquier tipo de cálculo matemático, ya esté orientado a ingeniería o no; y como herramienta valiosa de investigación y aprendizaje, la adquisición de su licencia es una inversión que todas las universidades deben considerar.*

- *La programación es una práctica que debe ser fomentada y realizada durante todo el transcurso de la carrera de Ingeniería Electrónica, ya que facilita la asimilación y creación de conocimientos en el programa, además de servir como herramienta de análisis y cómputo.*

## 8. SUGERENCIAS

De forma didáctica, se proponen las siguientes actividades para que los estudiantes investiguen acerca del procesamiento paralelo, el manejo de clústeres de cálculo y la programación en MATLAB:

- Programar un *script* que lea los archivos de una carpeta fija en el disco duro y extraiga los momentos invariantes de todas las imágenes allí encontradas. Completar una versión secuencial –no paralela- y tomar tiempos de ejecución. Segmentar cada una de estas extracciones como tareas independientes, y hacerlas correr en el clúster. Tomar tiempos en cada caso, y repetir la ejecución para clústeres con cantidades diferentes de trabajadores. No se necesita desinstalar y reinstalar todo el clúster, solo agregar o quitar trabajadores registrados con el mismo Job Manager. Analizar los resultados obtenidos. Adicionalmente, se pueden modificar las tareas para cambiar la granulosidad de la versión paralelizada y repetir la experiencia (o asignar un grado de granulosidad a cada grupo en el laboratorio).

- Aplicar una máscara de convolución de tamaño considerable (9x9 o más) a una imagen de alta resolución (de unos 7 a 12 MP) de forma secuencial –no paralela – y de forma distribuida (repartiendo los arreglos con la información de la imagen entre diferentes nodos del clúster). Tomar tiempos de ejecución mediante las funciones **`tic`** y **`toc`**, y comparar resultados. Toda la información referente a arreglos distribuidos se encuentra disponible en la documentación de la Toolbox de cálculo distribuido para MATLAB.

- Examinar los ejemplos disponibles en la sección **Distributed Computing** en el apartado **Demos** de la ayuda de MATLAB.

## GLOSARIO

**ADMINISTRADOR DE TRABAJO DE MATHWORKS (JOB MANAGER):** es el proceso de MathWorks que organiza los trabajos y asigna las tareas a los trabajadores. Un programa externo que realice este proceso, se conoce como despachador (scheduler). En general, el término "despachador" hace referencia al administrador de trabajo.

**APLICACIÓN DE PARTICULAS GRUESAS (COARSE-GRAINED APPLICATION):** es una aplicación cuyo tiempo de ejecución es significativamente mayor que el tiempo de comunicación necesario para iniciar y detener el programa. Estas aplicaciones también son llamadas aplicaciones desconcertantemente paralelas.

**APLICACIÓN DE PARTÍCULAS FINAS (FINE-GRAINED APPLICATION):** es una aplicación en la cual el tiempo de ejecución es significativamente menor que el tiempo de comunicación necesario para iniciar y detener el programa. Comparar con las aplicaciones de partículas gruesas.

**APLICACIÓN DISTRIBUIDA:** esta aplicación corre independientemente en varios nodos, posiblemente con diferentes parámetros de entrada. No hay comunicación, información compartida, o puntos de sincronización entre los nodos. Las aplicaciones distribuidas pueden ser de partículas gruesas o finas.

**APLICACIÓN PARALELA:** la misma aplicación que corre en varios labs (nodos) simultáneamente; con comunicación, datos compartidos, o puntos de sincronización entre labs.

**BASE DE DATOS DEL ADMINISTRADOR DE TRABAJO:** es la base de datos que usa el administrador de trabajo para almacenar la información acerca del trabajo y sus tareas.

**CHECKPOINTBASE:** es el nombre del parámetro en el archivo mdce_def que determina la localización del administrador de trabajos y los directorios de chequeo de los trabajadores.

**DESPACHADOR (SCHEDULER):** Es el proceso que organiza trabajos y asigna tareas a los trabajadores, ya sea de una empresa externa o el mismo administrador de trabajos de MathWorks.

**DIRECTORIO DE CHEQUEO (CHECKPOINT DIRECTORY):** lugar en donde se almacena la información de chequeo del administrador de trabajos y la información de chequeo del trabajador.

**CLIENTE, CLIENTE MATLAB:** es la sesión MATLAB que define y reparte el trabajo. Usualmente, esta es la sesión de MATLAB en la cual el programador desarrolla y modela las aplicaciones. También se conoce como el cliente MATLAB.

**COMPUTADORA CLIENTE:** es la computadora que ejecuta el cliente MATLAB.

**CLÚSTER:** es un conjunto de computadores conectados en red y destinados para un propósito común.

**CLÚSTER HETEROGÉNEO:** un clúster que no es homogéneo.

**CLÚSTER HOMOGÉNEO:** un clúster de máquinas idénticas, en términos tanto de hardware como de software.

**COMPUTADORA:** un sistema con uno o más procesadores.
**CÁLCULO DISTRIBUIDO:** es el cálculo con aplicaciones distribuidas, ejecutando la aplicación en varios nodos simultáneamente.
**DNS (DOMAIN NAME SYSTEM):** sistema que traduce nombres de dominio de Internet a direcciones IP.
**GRANULOSIDAD: es el** grado de segmentación de un problema entre los trabajadores disponibles, según este parámetro se puede crear aplicaciones de grano fino de grano grueso.
**INFORMACIÓN DE CHEQUEO (CHECKPOINT) DEL JOB MANAGER:** es un estado o instantánea con la información necesaria para que el Job Manager se recupere de un colapso o reinicio en el sistema.
**INFORMACIÓN DE CHEQUEO (CHECKPOINT) DEL WORKER:** archivos requeridos por el trabajador durante la ejecución de tareas.
**LABS:** cuando los trabajadores inician, por defecto trabajan de forma independiente. Cuando se conectan a los demás y trabajan como grupo se les conoce como labs.
**LOCALHOST:** nombre del host actual, es usado por las computadoras como nombre propio dentro de la red. Cada vez que una pc en la red hace referencia al localhost, está hablando de sí misma.
**LOGDIR:** parámetro que define el directorio de almacenado de registros (logs) en el archive mdce_def.
**MDCE_DEF:** archivo que define todas las opciones por defecto para los procesos del MDCE, permitiéndole ajustar las preferencias o definiciones con la forma de parámetros y valores.
**MDCE:** servicio que debe estarse ejecutando en todas las máquinas antes de que puedan correr como administrador de trabajo o trabajador. Este es el proceso origen del motor, asegurándose de que los procesos de administrador de trabajo y trabajador siempre estén ejecutándose. Note que todo el nombre del programa y del servicio es en minúsculas.
**MPI (Message Passing Interface):** interfaz de tránsito de mensajes, es el modo en el cual los labs se comunican entre sí mientras ejecutan tareas en un mismo trabajo.
**NODO:** computadora que hace parte de un clúster.
**NODO CABEZA:** usualmente es el nodo del clúster designado para ejecutar el administrador de trabajo y el administrador de licencia. Frecuentemente, es útil correr en una sola máquina todos los procesos no relacionados directamente al trabajo.
**OVERHEAD:** en comunicaciones, son los códigos adicionales transmitidos para controlar y revisar errores, lo cual acarrea más tiempo a procesar.
**PROCESO DE BÚSQUEDA DEL ADMINISTRADOR DE TRABAJO:** proceso que permite a los clientes, trabajadores y administradores de trabajo encontrarse entre sí. Inicia automáticamente cuando el administrador de trabajo se inicia.
**PUERTO ALEATORIO (RANDOM PORT):** puerto TCP sin privilegios; i.e., un puerto aleatorio TCP por encima de 1024.

**REGISTRAR WORKER:** *acción que ocurre cuando se inicia un Worker y un Job Manager, y el Worker contacta al Job Manager, identificándose e indicando que está listo para recibir tareas.*

**TAREA (TASK):** *segmento de un trabajo, a ser evaluado por un trabajador.*

**TOOLBOX:** *aditamento o adicional que se instala sobre el MATLAB – o que viene incluida, dependiendo de la licencia adquirida – el cual permite ampliar las capacidades del programa, permitiendo hacer cómputos especializados más avanzados y diferentes a los que vienen por defecto.*

**TRABAJO (JOB):** *es la operación completa a gran escala, compuesta de un conjunto de tareas, a realizar en MATLAB.*

**TRABAJADOR (WORKER):** *es el proceso de MATLAB que realiza los cálculos de la tarea. También conocido como el trabajador MATLAB, proceso trabajador, o simplemente worker.*

## BIBLIOGRAFÍA

### Vínculos Web (Internet):


Sitio oficial de MathWorks. [en línea]. oct-dic 2006, [citado el 23 de octubre 2006] <avaliable from Internet: www.mathworks.com >.

Motor de búsqueda. [en línea]. Ene-jun 2007, [citado el 5 febrero 2007] <avaliable from Internet: www.google.com>.

Enciclopedia universal. [en línea]. oct 2006-jun 2007, [citado el 23 octubre 2006] <avaliable from Internet: www.wikipedia.com/es>.

Revista de divulgación sobre ciencias. [en línea]. oct-dic 2006, [citado el 6 noviembre] <avaliable from Internet: www.bornet.es>.

Publicación de tecnologías e informática. [en línea]. oct-nov 2006, [citado el 6 noviembre 2006] <avaliable from Internet: www.computerworld.com.co >.

Manual técnico clúster IDEAM. [en línea]. nov-dic 2006, [citado el 12 noviembre 2006] < avaliable from Internet: www.mdk.ideam.gov.co >.

Grupos reconocidos de colciencias. [en línea]. oct-dic 2006, [citado el 30 octubre 2006] <avaliable from Internet: www.zulia.colciencias.gov.co>.

Universidad autónoma de Manizales. [en línea]. oct-nov 2006, [citado el 13 noviembre 2006] <avaliable from Internet: www.computerworld.com.co >.

Proyecto de eficiencia y paralelización de algoritmos UN. [en línea]. oct-dic 2006, [citado el 19 noviembre 2006] <avaliable from Internet: www.exodus.physics.ucla.edu>.

Implementación de clúster beowulf como firewall. [en línea]. oct-dic 2006, [citado el 20 noviembre 2006] <avaliable from Internet: www.acis.org.co >.

# ANEXOS

## ANEXO A. ALGORITMO DE EJEMPLO, VERSION NO PARALELA

### Archivo "withdear.m"

```
%programa de Cálculo de rutas
%se etiquetará las operaciones q son independientes de otro proceso o
%resultado, como también las dependencias..
%cuadro   %proceso indp //grafica el terreno(mapa)
%//vectores iniciales de caminos
line1=[3*ones(1,24) 5*ones(1,24)];
line2=[3*ones(1,6) 5*ones(1,6) 4*ones(1,6) 3*ones(1,6) 5*ones(1,6)
4*ones(1,18)];
line3=4*ones(1,48);
line4=[5*ones(1,12) 2*ones(1,6) 4*ones(1,12) 2*ones(1,6) 5*ones(1,12)];
line5=[5*ones(1,24) 3*ones(1,24)];      %bloque indp //5 rutas de esquina
a esquina
load opcionespath   %carga optgaX //3 esquemas de mutación
%los siguientes procesos son indp pero necesitan q ya se halla ejecutado
%rutas y cargado optga...
%//rutas para optga1
optga1.options.PlotInterval=500
optga2.options.PlotInterval=500
optga3.options.PlotInterval=500
try
    optga1.options.InitialPopulation=line1;
    [x v]=ga(optga1);
    path1(1,:)=round(x);
    %
    optga1.options.InitialPopulation=line2;
    [x v]=ga(optga1);
    path1(2,:)=round(x);
    %
    optga1.options.InitialPopulation=line3;
    [x v]=ga(optga1);
    path1(3,:)=round(x);
    %
    optga1.options.InitialPopulation=line4;
    [x v]=ga(optga1);
    path1(4,:)=round(x);
    %
    optga1.options.InitialPopulation=line5;
    [x v]=ga(optga1);
    path1(5,:)=round(x);
    %
    optga2.options.InitialPopulation=line1;
    [x v]=ga(optga2);
    path2(1,:)=round(x);
    %
```

```
    optga2.options.InitialPopulation=line2;
    [x v]=ga(optga2);
    path2(2,:)=round(x);
    optga2.options.InitialPopulation=line3;
    [x v]=ga(optga2);
    path2(3,:)=round(x);
    %
    optga2.options.InitialPopulation=line4;
    [x v]=ga(optga2);
    path2(4,:)=round(x);
    %
    optga2.options.InitialPopulation=line5;
    [x v]=ga(optga2);
    path2(5,:)=round(x);
    %
    optga3.options.InitialPopulation=line1;
    [x v]=ga(optga3);
    path3(1,:)=round(x);
    %
    optga3.options.InitialPopulation=line2;
    [x v]=ga(optga3);
    path3(2,:)=round(x);
    %
    optga3.options.InitialPopulation=line3;
    [x v]=ga(optga3);
    path3(3,:)=round(x);
    %
    optga3.options.InitialPopulation=line4;
    [x v]=ga(optga3);
    path3(4,:)=round(x);
    %
    optga3.options.InitialPopulation=line5;
    [x v]=ga(optga3);
    path3(5,:)=round(x);
end
%
%//visualización...
%cada esquema depende de su pathX y son independientes entres si..
npath='primer esquema';
cuadro
for i=1:5
    draw_path(path1(i,:),rand(1,3),i/6);
end
%
npath='segundo esquema';
cuadro
for i=1:5
    draw_path(path2(i,:),rand(1,3),i/6);
end
%
npath='tercer esquema';
cuadro
```

```
for i=1:5
    draw_path(path3(i,:),rand(1,3),i/6);
end
```

### Archivo "camino.m"

```
%devuelve un valor proporcional según la dificultad del terreno y ...
%distancia contenido en mapa (terreno.mat)
function z=camino(x)
load terreno
%load mmp
n=25;    %dimensión de mapa, cambiar si el ejemplo cambia
i=1;j=1;z=0;
%nomenclatura
%   1  2
%   | /
%   x -3
%   | \
%   5  4
for k=1:length(x)
    switch round(x(k))
        case 1
            z=z+2;
            j=j-1;
            if j<1
                z=7500;
                return;
            end
        case 2
            z=z+3;
            j=j-1;
            i=i+1;
            if (j<1)||(i>n)
                z=7500;
                return;
            end
        case 3
            z=z+2;
            i=i+1;
            if i>n
                z=7500;
                return;
            end
        case 4
            z=z+3;
            i=i+1;
            j=j+1;
            if (j>n)||(i>n)
                z=7500;
                return;
            end
```

```
        otherwise
            z=z+2;
            j=j+1;
            if j>n
                z=7500;
                return;
            end
    end
    z=z+mapa(j,i);
    if (i==n)&&(j==n)
        break;
    end
end
if (i~=n)||(j~=n)
    z=7500;
end
```

### Archivo "cuadro.m"

```
%dibuja a mapa contenido en terreno.mat
%mapa debe ser cuadrado de dimensión max 25x25
figure('Name',['Caminos calculados para ' npath],'NumberTitle','off')
axis([0 27 0 27]);
axis off
load terreno
%recordad que mapa es una matriz cuadrada max 25x25, con celdas de valor
0, 10, 20, 30
%que representan la dificultad del terreno respectivamente...
sz=size(mapa);
hold on
for i=1:sz(2)
    for j=1:sz(1)
        switch mapa(j,i)
            case 0
                clr=[1 1 1];
            case 10
                clr=[0.66 0.66 0.66];
            case 20
                clr=[0.33 0.33 0.33];
            otherwise
                clr=[0 0 0];
        end
        fill([i,i+1,i+1,i,i],[27-j,27-j,26-j,26-j,27-j],clr);
    end
end
hold off
```

### Archivo "draw_path.m"

```
function draw_path(ruta,tono,offs)
n=25;   %dimensión de mapa, cambiar si el ejemplo cambia
m=length(ruta);
i=[1 1];j=[1 1];k=1;
hold on
while(i(2)~=n || j(2)~=n)
    if k>m
        break;
    end
    i(1)=i(2);
    j(1)=j(2);
    switch ruta(k)
        case 1
            j(2)=j(2)-1;
        case 2
            i(2)=i(2)+1;
            j(2)=j(2)-1;
        case 3
            i(2)=i(2)+1;
        case 4
            i(2)=i(2)+1;
            j(2)=j(2)+1;
        otherwise
            j(2)=j(2)+1;
    end
    line(i+0.3+offs/2,27-j-offs,'color',tono);
    k=k+1;
end
hold off
```

## ANEXO B. ALGORITMO DE EJEMPLO, VERSION PARALELA

### Archivo "withdear_par.m"

*(Los demás archivos permanecen iguales)*

```
%programa de Cálculo de rutas
%se etiquetará las operaciones q son independientes de otro proceso o
%resultado, como también las dependencias..
%cuadro   %proceso indp //grafica el terreno(mapa)
%//vectores iniciales de caminos
tic
% CONDICIONES INICIALES.
% Son 5 rutas. Cada una de la esquina superior izquierda a la inferior
% derecha.
line1=[3*ones(1,24) 5*ones(1,24)];
line2=[3*ones(1,6) 5*ones(1,6) 4*ones(1,6) 3*ones(1,6) 5*ones(1,6)
4*ones(1,18)];
line3=4*ones(1,48);
line4=[5*ones(1,12) 2*ones(1,6) 4*ones(1,12) 2*ones(1,6) 5*ones(1,12)];
line5=[5*ones(1,24) 3*ones(1,24)];

load opcionespath   %carga optgaX //3 esquemas de mutación
%los siguientes procesos son indp pero necesitan q ya se halla ejecutado
%rutas y cargado optga...


% Ahora deberá evaluarse cada uno de los 3 esquemas de mutación con las 5
% condiciones iniciales, de esta forma obtendremos 15 soluciones
% optimizadas para este problema

%#########################################################
%                MODIFICACION PARALELA
%
JOBMAN='JOBMAN1';
JOBMANHOST='jarl';

% Se busca un job manager
jm =
findResource('scheduler','type','jobmanager','name',JOBMAN,'LookupURL',JO
BMANHOST);

% Se crea un job, un trabajo
mijob = createJob(jm,'Name','Extraccion');


% Se configuran los archivos de dependencias necesarios para correr las
% tareas
set(mijob,'FileDependencies',{'camino.m','terreno.mat'})
```

```
tareas={}

optga1.options.InitialPopulation=line1;
% [x v]=ga(optga1);
% x es el vector respuesta
% v es el valor de la función para ese vector
tareas{numel(tareas)+1}=createTask(mijob, @ga, 2, {optga1});
%path1(1,:)=round(x);

optga1.options.InitialPopulation=line2;
%[x v]=ga(optga1);
tareas{numel(tareas)+1}=createTask(mijob, @ga, 2, {optga1});
%path1(2,:)=round(x);

optga1.options.InitialPopulation=line3;
%[x v]=ga(optga1);
tareas{numel(tareas)+1}=createTask(mijob, @ga, 2, {optga1});
%path1(3,:)=round(x);

optga1.options.InitialPopulation=line4;
%[x v]=ga(optga1);
tareas{numel(tareas)+1}=createTask(mijob, @ga, 2, {optga1});
%path1(4,:)=round(x);

optga1.options.InitialPopulation=line5;
%[x v]=ga(optga1);
tareas{numel(tareas)+1}=createTask(mijob, @ga, 2, {optga1});
%path1(5,:)=round(x);

optga2.options.InitialPopulation=line1;
%[x v]=ga(optga2);
tareas{numel(tareas)+1}=createTask(mijob, @ga, 2, {optga2});
%path2(1,:)=round(x);

optga2.options.InitialPopulation=line2;
%[x v]=ga(optga2);
tareas{numel(tareas)+1}=createTask(mijob, @ga, 2, {optga2});
%path2(2,:)=round(x);

optga2.options.InitialPopulation=line3;
%[x v]=ga(optga2);
tareas{numel(tareas)+1}=createTask(mijob, @ga, 2, {optga2});
%path2(3,:)=round(x);

optga2.options.InitialPopulation=line4;
%[x v]=ga(optga2);
tareas{numel(tareas)+1}=createTask(mijob, @ga, 2, {optga2});
%path2(4,:)=round(x);

optga2.options.InitialPopulation=line5;
```

```
%[x v]=ga(optga2);
tareas{numel(tareas)+1}=createTask(mijob, @ga, 2, {optga2});
%path2(5,:)=round(x);

optga3.options.InitialPopulation=line1;
%[x v]=ga(optga3);
tareas{numel(tareas)+1}=createTask(mijob, @ga, 2, {optga3});
%path3(1,:)=round(x);

optga3.options.InitialPopulation=line2;
%[x v]=ga(optga3);
tareas{numel(tareas)+1}=createTask(mijob, @ga, 2, {optga3});
%path3(2,:)=round(x);

optga3.options.InitialPopulation=line3;
%[x v]=ga(optga3);
tareas{numel(tareas)+1}=createTask(mijob, @ga, 2, {optga3});
%path3(3,:)=round(x);

optga3.options.InitialPopulation=line4;
%[x v]=ga(optga3);
tareas{numel(tareas)+1}=createTask(mijob, @ga, 2, {optga3});
%path3(4,:)=round(x);

optga3.options.InitialPopulation=line5;
%[x v]=ga(optga3);
tareas{numel(tareas)+1}=createTask(mijob, @ga, 2, {optga3});
%path3(5,:)=round(x);


disp('Subiendo trabajo')
submit(mijob);

disp('ejecutando trabajo')
waitForState(mijob);

disp('Extrayendo resultados')
results=getAllOutputArguments(mijob);

for i=1:5
    x=results{i,1};
    v=results{i,2};
    path1(i,:)=round(x);
end

for i=6:10
    x=results{i,1};
    v=results{i,2};
    path2(i-5,:)=round(x);
end
```

```
for i=11:15
    x=results{i,1};
    v=results{i,2};
    path3(i-10,:)=round(x);
end


%//visualización...
%cada esquema depende de su pathX y son independientes entres si..
npath='primer esquema';
cuadro;
for i=1:5
    draw_path(path1(i,:),rand(1,3),i/6);
end
%
npath='segundo esquema';
cuadro;
for i=1:5
    draw_path(path2(i,:),rand(1,3),i/6);
end
%
npath='tercer esquema';
cuadro;
for i=1:5
    draw_path(path3(i,:),rand(1,3),i/6);
end

toc
```