U NIVERSIDAD *A* UTÓNOMA
M ETROPOLITA NA

DIVISIÓN DE CIENCIAS BASICAS E INGENIERIA

INGENIERIA EN COMPUTACIÓN

NOMBRE DEL PROYECTO:

Monitoreo de Daños a Estructuras Civiles en Diferentes Sistemas Distribuidos

NOMBRE DEL ASESOR:
Dra. Silvia Beatriz González Brambila

NOMBRE DEL ALUMNO:
Ricardo Haro García

MATRICULA:
205202154

FECHA:
Diciembre 2009

Índice

1. INTRODUCCIÓN

Actualmente surge la necesidad de resolver problemas cada vez más complejos, que son inabordables con la tecnología y capacidad de una sola computadora personal debido a que es inaceptable el tiempo de espera, una solución es utilizar grandes cantidades de recursos computacionales (Hardware) lo cual podría ser muy costoso.

Es por eso que desde hace algunos años las necesidades de cómputo de numerosas aplicaciones obligan a desarrollar software eficiente y seguro para plataformas multiprocesador. Además, el auge de los procesadores multinúcleo y de las redes de computadoras ha aumentado la difusión del procesamiento paralelo que cada vez está más al alcance del público en general. No obstante, para utilizar los sistemas paralelos y/o distribuidos de forma eficiente es necesaria la programación paralela.

Por lo anterior en este proyecto se hablara de la ejecución remota o programación paralela, haciendo énfasis en *MPI[1]* y *TREx[2]* este último no tiene la trayectoria que tiene *MPI,* es por eso que se presenta una serie de pruebas y experimentos que permiten comparar ambos sistemas, con el objetivo de contar con elementos para definir cuál es mejor en determinadas condiciones, que se presentan más adelante.

En la República Mexicana se han presentado muchos fenómenos naturales que han afectado las estructuras con resultados catastróficos. Fenómenos naturales como sismos, huracanes, temporales de lluvias, etc. Estos han provocado daño significativo en las estructuras.

Dentro de las obras civiles más importantes, se pueden mencionar las vías de comunicación (puentes, viaductos, túneles, etc.), los sistemas de aprovechamientos hidráulicos, los edificios de servicios públicos (hospitales, estaciones de bomberos, etc.) que brindan servicio a toda la población.

Debido a la creciente complejidad de modelos científicos aplicados a la simulación del desgaste de las estructuras, en general, a menudo las herramientas de simulación a gran escala exigen una gran potencia de cálculo para producir resultados en un tiempo razonable. Por ejemplo, los sistemas para simular terremotos, la búsqueda de petróleo, el simple desgaste de estructuras civiles y las simulaciones graficas.

Una de las soluciones a este problema es utilizar, el procesamiento paralelo, pero tiene demasiadas limitantes, una de ellas es que no está al alcance de cualquier usuario, debido a su costo elevado.

Otra de las soluciones es hacer uso de la ejecución remota mediante un sistema distribuido, esta solución puede ser la más accesible para cualquier usuario que requiera procesar una gran cantidad de cálculos.

2. OBJETIVOS

Implementar de manera paralela, en dos sistemas diferentes el modelo MAC[1] capaz de detectar daños que no sean localizados a simple vista en diferentes tipos de estructuras utilizadas en la construcción.

Con la ayuda del algoritmo MAC podremos definir entre dos sistemas cual brinda un mejor desempeño en determinadas situaciones, y al hablar de desempeño me refiero a disminuir tiempos de ejecución.

Los sistemas que fueron seleccionados para someterlos a pruebas son TREx y MPI.

Se investigó primordialmente el funcionamiento y características de cada uno de los sistemas, para después con la ayuda de las herramientas que proporciona cada uno implementar un algoritmo basado en el método MAC, añadiendo para el caso de MPI las bibliotecas e instrucciones necesarias para realizar ejecución remota utilizando una distribución Linux. En el caso de TREx se usó el mismo algoritmo basado en el método MAC pero será ejecutado de manera remota desde la plataforma de Windows, cabe mencionar que TREx no necesita usar librerías ajenas al código original, simplemente se necesita el ejecutable de MAC y llevar a cabo una buena configuración para su ejecución remota, para ambos casos se implementaran en lenguaje C.

Se elegirá un programa desarrollado en MPI para probar su funcionalidad correcta, y se modificara para ejecutarlo con TREx para realizar comparaciones entre estos sistemas.

Finalmente se realizaran las pruebas necesarias para determinar cuál de los dos sistemas redujo más el tiempo de ejecución total.

---

[1] *Transparent Remote Execution: Sistema de distribución de carga entre procesadores experimental de la Universidad de California*

[2] *Message Passing Interface Es un estándar que define la sintaxis y la semántica de las funciones contenidas en una biblioteca de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores*

3. MAC

Se implementó en lenguaje C el algoritmo de MAC que es capaz de calcular el deterioro improvisado de alguna estructura civil, este código se adaptó al formato de MPI.

El método MAC muestra una correlación entre los dos estados de la estructura (estructura sin daño y con daño). Por lo que se tendrán dos archivos de entrada, el primer archivo será con los datos de la estructura nueva, es decir, en su estado original sin ningún tipo de daño, en el segundo archivo se simulara daño severo en la estructura.

El tipo de dato que se manejara en estos archivos es de tipo flotante donde el rango es de (0,1).

Este algoritmo es muy sensible a pequeñas variaciones en rigideces en los elementos estructurales.

El MAC es definido como un escalar constante que relaciona el grado de consistencia (linealmente) entre un modo i y otro de modo s, entendiendo a modo i como la condición inicial de la estructura (sin defectos) y el modo s como el estado actual/final de la estructura (con o sin defectos).

El método MAC consiste en la siguiente función:

$$MAC_{(i,j)} = \frac{\left(\{\phi\}_{id}{}^{T}\{\phi\}_{j_u}\right)^2}{\{\phi\}_{id}{}^{T}\{\phi\}_{id}{}^{T} \times \{\phi\}_{j_u}{}^{T}\{\phi\}_{j_u}}$$

Donde:

Φid es un vector de tipo flotante que corresponde al estado óptimo de la estructura y este vector fue obtenido mediante sensores conectados a una estructura y almacenados en una de las matrices capturadas.

φju es un vector de tipo flotante que corresponde al último estado de la estructura (con o sin defectos) almacenado en otra de las matrices capturadas.

T se refiere a la transpuesta del vector correspondiente.

Los valores calculados con el MAC varían entre cero y uno. Valores pequeños indican baja correlación entre los vectores, mientras que valores grandes indican una correlación alta entre ellos, es decir si el valor de MAC tiende a cero (0) existe daño en la estructura, si el valor tiende a uno (1) la estructura no tiene daños considerables.

Implementado el algoritmo anterior, dará como resultado (salida) un número escalar que estará en el rango antes mencionado y se podrá definir si la estructura está dañada o no.

4. MPI

MPI (por sus siglas en inglés *Message Passing Interface*) es un modelo de comunicación ampliamente usado en computación paralela.

MPI es un estándar que ayuda a la implementación de sistemas paralelos utilizando pase de mensajes, diseñado por un grupo de investigadores para que pueda funcionar en una amplia variedad de computadoras paralelas y de forma tal que los códigos sean portables.

Su diseño está inspirado en máquinas con una arquitectura de memoria distribuida (véase figura 1) en donde cada procesador es propietario de cierta memoria y la única forma de intercambiar información es a través de mensajes, sin embargo, hoy en día también encontramos implementaciones de MPI en máquinas de memoria compartida aun que es muy poco frecuente (véase figura 2).

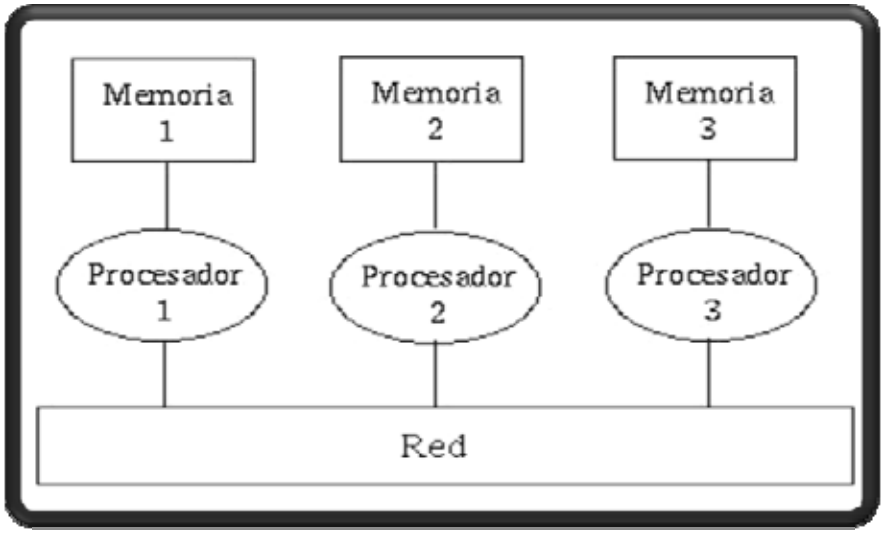

Figura 1: Memoria Distribuida, donde cada Procesador posee un cierto tamaño de memoria

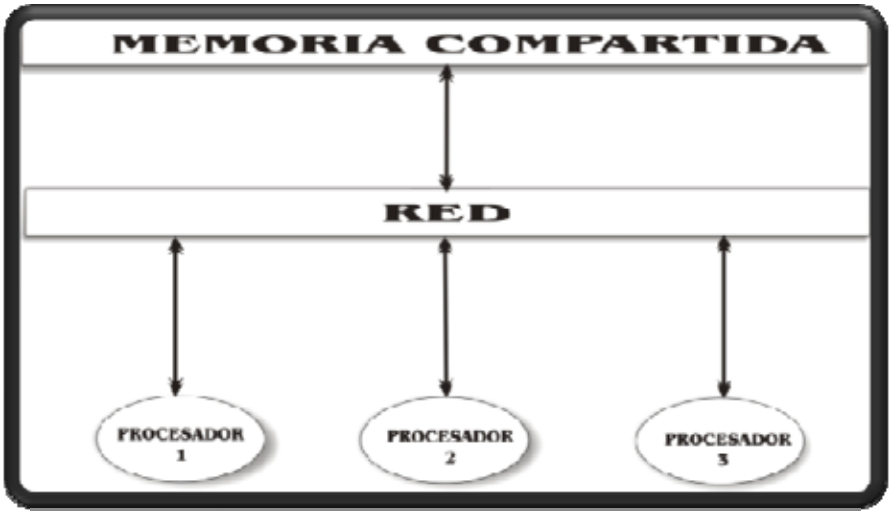

Figura 2: Muestra Memoria Compartida por un cierto número de procesadores

MPI atiende a una estructura SIMD[3] que consisten en instrucciones que aplican una misma operación sobre un conjunto grande de datos.

Todos los procesadores reciben la misma instrucción de la unidad de control, pero operan sobre diferentes conjuntos de datos (véase figura 3). Es decir la misma instrucción es ejecutada de manera síncrona por todas las unidades de procesamiento.

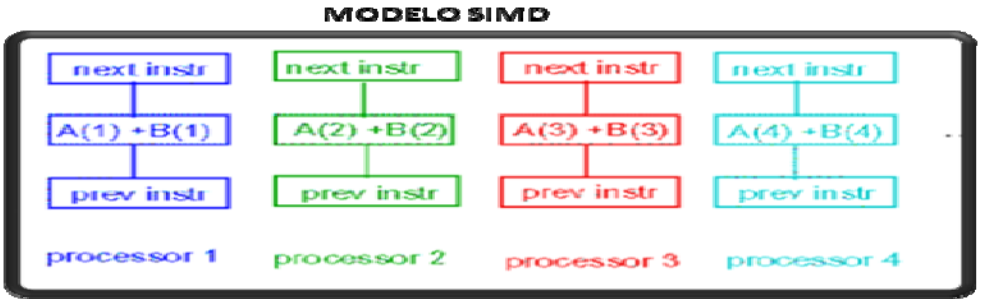

Figura 3: Muestra el modelo SIMD, que nos dice que todos los procesadores realizan la misma operación pero con diferentes datos

[3] SIMD (Single Instruction, Multiple Data) Una Instrucción, Múltiples Datos es una técnica empleada para conseguir paralelismo a nivel de datos.

4.1 OBJETIVO DE MPI

El objetivo principal de MPI es lograr la portabilidad del código fuente a través de diferentes máquinas, tratando de obtener un grado de portabilidad comparable al de un lenguaje de programación que permita ejecutar remotamente, de manera transparente, aplicaciones sobre sistemas heterogéneos; objetivo que no debe ser logrado a expensas del rendimiento. De manera simple, el objetivo de MPI es desarrollar un estándar para ser ampliamente usado que permita escribir programas usando pase de mensajes y así lograr el paralelismo. Por lo tanto, la interface debería establecer un estándar práctico, portable, eficiente y flexible.

4.2 CARACTERISTICAS PRINCIPALES DE MPI

- Dirigido a proveer portabilidad del código fuente.
- Ofrece gran funcionalidad
- Soporta gran cantidad de tipo de datos
- Soporta datos definidos por el usuario
- Maneja diferentes topologías
- El estándar es extenso en cuanto al número de rutinas que se especifican; contiene alrededor de 129 funciones muchas de la cuales tienen numerosas variantes y parámetros. Esto no implica una relación directamente proporcional entre la complejidad y el número de funciones; muchos programas paralelos pueden ser escritos usando sólo 6 funciones básicas.
- Con MPI el número de procesos requeridos se asigna antes de la ejecución del programa, y no se crean procesos adicionales mientras la aplicación se ejecuta. A cada proceso se le asigna una variable que se denomina rank, la cual identifica a cada proceso, en el rango de 0 a p-1, donde p es el número total de procesos. El control de la ejecución del programa se realiza mediante la variable rank; que permite determinar qué proceso ejecuta determinada porción de código. En MPI se define un “comunicador” como una colección de procesos, los cuales pueden enviar mensajes el uno al otro.

En la figura (4) se muestra un diagrama a bloques del programa con MPI.

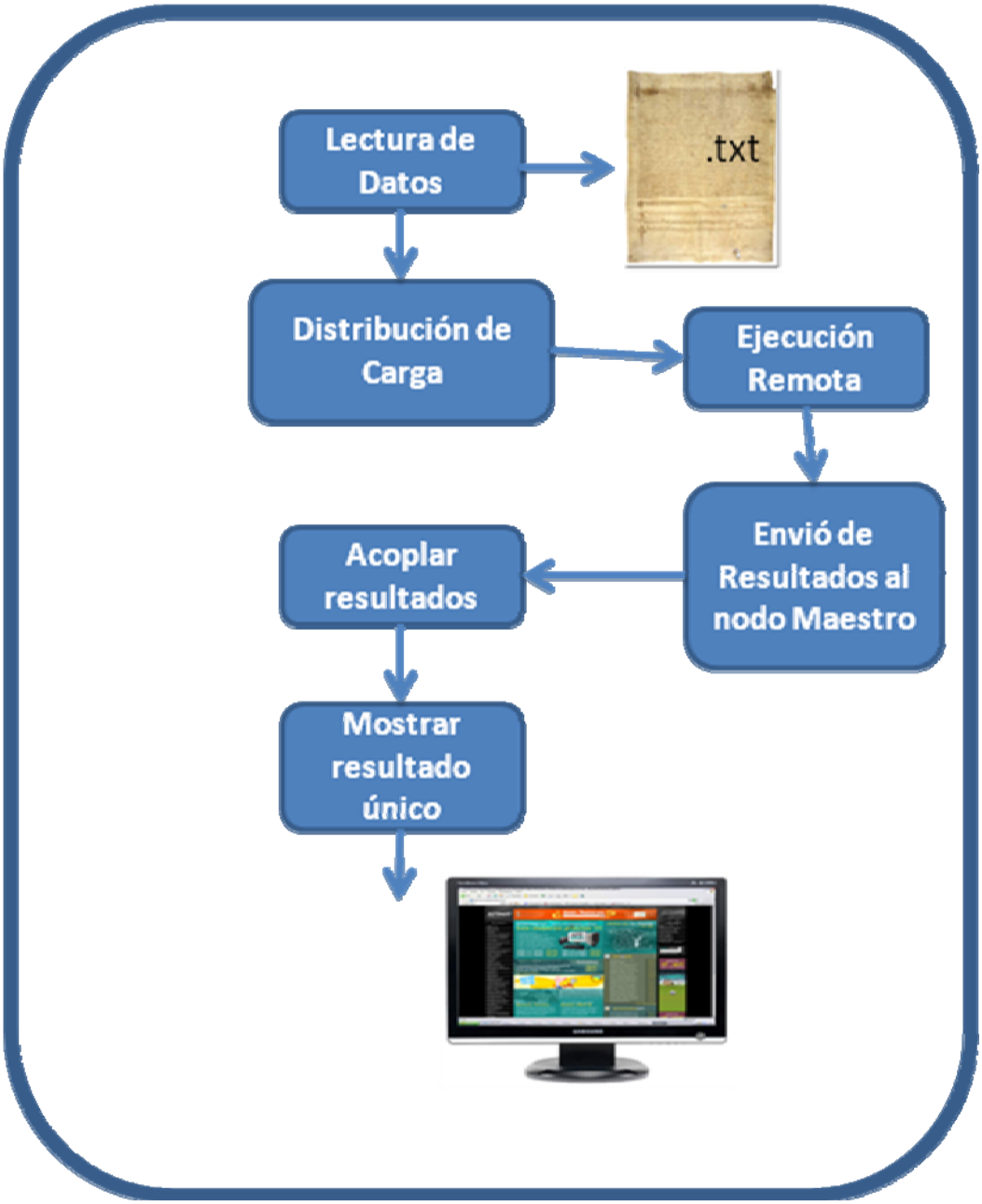

Figura 4: *Diagrama a bloques correspondiente al funcionamiento de MPI con el algoritmo basado en MAC*

4.3 CUÁNDO USAR MPI

- Si la portabilidad es necesaria.
- Si se requiere una aplicación paralela portable.
- La interrelación de los datos es dinámica o irregular y no se ajusta a un modelo de datos paralelos. MPI, y en general el esquema de pase de mensajes, facilitan la programación en estos casos.
- Necesidad de crear aplicaciones que necesiten una gran potencia de cálculo.
- Resolver problemas inabordables con un solo procesador.
- Alcanzar alto desarrollo de programación paralela.

4.4 ELEMENTOS BÁSICOS DE LA PROGRAMACIÓN CON MPI

Las funciones de MPI se dividen en cuatro clases:

1.Llamadas utilizadas para inicializar, administrar y finalizar comunicaciones:

Esta clase de llamadas permite inicializar la biblioteca de paso de mensajes, identificar el número de procesos (size) y el rango de los procesos (rank).

MPI_Init(): funciona para iniciar la aplicación paralela.
int MPI_Init(int *argc, char ***argv);

MPI_Comm_size(): sirve para averiguar el número de procesos que participan en la aplicación.
int MPI_Comm_size (MPI_Comm comm, int *size);

MPI_Comm_rank(): sirve para que cada proceso averigüe su dirección (identificador) dentro de la colección de procesos que componen la aplicación.
int MPI_Comm_rank (MPI_Comm comm, int *rank);

MPI_Finalize(): sirve para dar por finalizada la aplicación.
int MPI_Finalize(void);

2. Llamadas utilizadas para transferir datos entre un par de procesos:

Estas llamadas incluyen operaciones de comunicación punto a punto, para diferentes tipos de actividades de envío y recepción.

MPI_Send(): permite enviar información desde un proceso a otro.

MPI_Recv(): permite recibir información desde otro proceso.

MPI_Wait(): es una llamada bloqueante y retorna cuando la operación de envío o recepción se completa.

MPI_Test(): permite verificar si la operación de envío o recepción ha finalizado.

3. Llamadas para transferir datos entre varios procesos:

Estas llamadas son conocidas como operaciones grupales y proveen operaciones de comunicaciones entre grupos de procesos.

MPI_Bcast(): permite a un proceso enviar una copia de sus datos a otros procesos dentro de un grupo definido por un comunicador.

MPI_Scatter(): establece una operación de distribución, en la cual un dato (arreglo de algún tipo de datos) se distribuye en diferentes procesos.

MPI_Gather(): establece una operación de recolección, en la cual los datos son recolectados en un sólo proceso.

MPI_Reduce(): permite que el proceso raíz recolecte datos desde otros procesos en un grupo, y los combine en un solo ítem de datos.

4. Llamadas utilizadas para crear tipos de datos definidos por el usuario: La última clase de llamadas provee flexibilidad en la construcción de estructuras de datos complejos.

MPI_Type_struct: sirve para definir un nuevo tipo de dato.

MPI_Pack: para empaquetar los datos.

4.5 PARTES DEL MENSAJE EN MPI

Por otro lado tenemos al mensaje, que es la manera en que se comunicaran los procesos y en la que los datos son enviados, su envoltura indica el proceso fuente y el destino.

El cuerpo del mensaje en MPI se conforma por tres piezas de información: buffer, tipo de dato y count.

El buffer, es la localidad de memoria donde se encuentran los datos de salida o donde se almacenan los datos de entrada.

El tipo de dato, indica el tipo de los datos que se envían en el mensaje.

La envoltura de un mensaje en MPI típicamente contiene la dirección destino, la dirección de la fuente y cualquier otra información que se necesite para transmitir y entregar el mensaje. Más específicamente la envoltura de un mensaje en MPI, consta de cuatro partes:

1. La fuente: identifica al proceso transmisor.

2. El destino: identifica al proceso receptor.

3. El comunicador: especifica el grupo de procesos a los cuales pertenecen la fuente y el destino.

4. La etiqueta (*tag*): permite clasificar el mensaje, que es un entero definido por el usuario que puede ser utilizado para distinguir los mensajes que recibe un proceso.

Y mediante estos mensajes se logra la comunicación y sincronización de procesos entre los diversos procesadores.

4.6 PASOS PARA EL FUNCIONAMIENTO ÓPTIMO DE MPI

Cada uno de los pasos que se mencionaran a continuación fueron probados en una distribución de Linux Ubuntu 8.10 Desktop Edition.

1. Instalar el paquete lam-runtime desde consola o desde la herramienta de instalaciones y actualizaciones *Synaptic[4]*, lam-runtime es un sistema de desarrollo para procesamiento en paralelo que trabaja sobre redes formadas por procesadores independientes utilizando programación estándar de paso de mensajes (MPI)

Synaptic[4] Es una interfaz gráfica para apt, el sistema de gestión de paquetes de Ubuntu
LAM[5] Local Area Multicomputer. Es un entorno de desarrollo de procesamiento paralelo ppensado para redes de trabajo con computadores independientes.

LAM[5] proporciona al usuario una librería API para el paso de mensajes entre diferentes nodos que colaboran para la realización de una aplicación paralela. Funciona en una gran cantidad de plataformas UNIX y permite trabajar con diferentes tipos de maquinas mezclados.

Desde consola se instala con el siguiente comando:

$ sudo apt-get install run-time

2. Se debe instalar la biblioteca contenida en el paquete *libc4-dev* que contiene mpi.h necesaria para la compilación satisfactoria del de algún programa desarrollado con algunas de las funciones a MPI.

Desde consola se instala con el siguiente comando:

$ sudo apt-get install libc4-dev

3. Una vez que se tiene el código fuente es necesario compilarlo esto se realiza con el siguiente comando:

$ mpicc –o nombre nombre.c

4. Cuando la compilación ha sido satisfactoria, se puede probar localmente su funcionamiento, esto se logra simplemente ejecutándolo, esto se hace con la instrucción mpirun, que ejecuta todos aquellos programas MPI.

Desde consola se uso mpirun bajo la siguiente sintaxis:

$mpirun –np 4 nombre

Donde np = numero de procesos a realizar.

5. Cuando se haya probado localmente o se quiera probar directamente con ejecución remota el siguiente paso es dar de alta los nodos que participaran en la ejecución, es decir se deben registrar las direcciones IP de cada uno de los nodos esclavos al nodo que fungirá como el nodo maestro o nodo servidor, las direcciones de los esclavos se incluyen en el archivo bhost.def este archivo se localiza en la carpeta /etc/lam/ y para poder configurarlo se debe iniciar una terminal como súper-usuario.

LAM[5] Local Area Multicomputer. Es un entorno de desarrollo de procesamiento paralelo ppensado para redes de trabajo con computadores independientes.

El archivo bhost.def inicialmente solo contiene:

localhost

y finalmente el archivo bhost.def del servidor quedara con todas las direcciones IP de los esclavos.

192.40.30.10
192.40.30.11
192.40.30.12
192.40.30.13
192.40.30.14

Y asi sucesivamente hasta incluir todos los esclavos que se encuentren dentro del rango de la LAN.

Todos los nodos esclavos también tienen un archivo bhost.conf que se encuentra en la misma ubicación que en la del servidor, en este caso solo se incluirá en el archivo la dirección IP del nodo maestro o servidor.

6. Posteriormente se lanzara el comando “recon” que reconocerá todos aquellos equipos registrados en el archivo bhost.def.
   Una vez que se lance este comando, el servidor pedirá las contraseñas necesarias para poder establecer una conexión segura entre nodos esclavo y el mismo servidor, esta contraseña será la original de cada equipo o la configurada por el administrador.

7. Cuando se haya hecho un canal seguro de comunicación entre servidor y los esclavos necesitamos portar LAM a todos los nodos, es decir lanzar demonios de control de proceso, control de entorno y paso de mensajes en cada nodo para crear entorno LAM/MPI y poder usar satisfactoriamente aplicaciones MPI.

Todo esto se logra con el comando:

$ lamboot

Durante la ejecución de este comando solicitara de nueva cuenta las contraseñas de cada uno de los nodos esclavos.
Si se tiene éxito en todas las validaciones de contraseñas se abrirá un canal de comunicación seguro y aparecerá en pantalla un mensaje de “Éxito”.

LOCALHOST. Es un nombre reservado que tienen todas las computadoras para referirse a sí mismo

8. Por último, para iniciar la ejecución remota se debe teclear desde el nodo servidor el comando:

$ lamexec –np 4 nombre

O de igual forma con el comando:

$ mpirun –np 4 nombre

Donde nombre pertenece al nombre de nuestro archivo “ejecutable”

Cabe mencionar que los archivos resultantes de la compilación como los archivos de entrada (si es que son necesarios), tendrán que estar en cada uno de los nodos esclavos en la carpeta personal del usuario creada por default durante la instalación de Ubuntu.

4.7 PRUEBAS REALIZADAS CON MPI

Una vez implementado el algoritmo MAC se realizaron varias pruebas en forma stand-alone y en ejecución remota para poder observar la eficiencia en cuestión de tiempo. Para realizar las pruebas nuestro programa necesita de dos archivos de entrada que como se mencionó anteriormente cada uno contendrá una matriz de tipo flotante de tamaño 5000x1000 (TamMatrix[5000][1000]).(En la Figura 5 se muestra parte de este archivo). Los archivos de entrada en promedio pesan 127Mb.

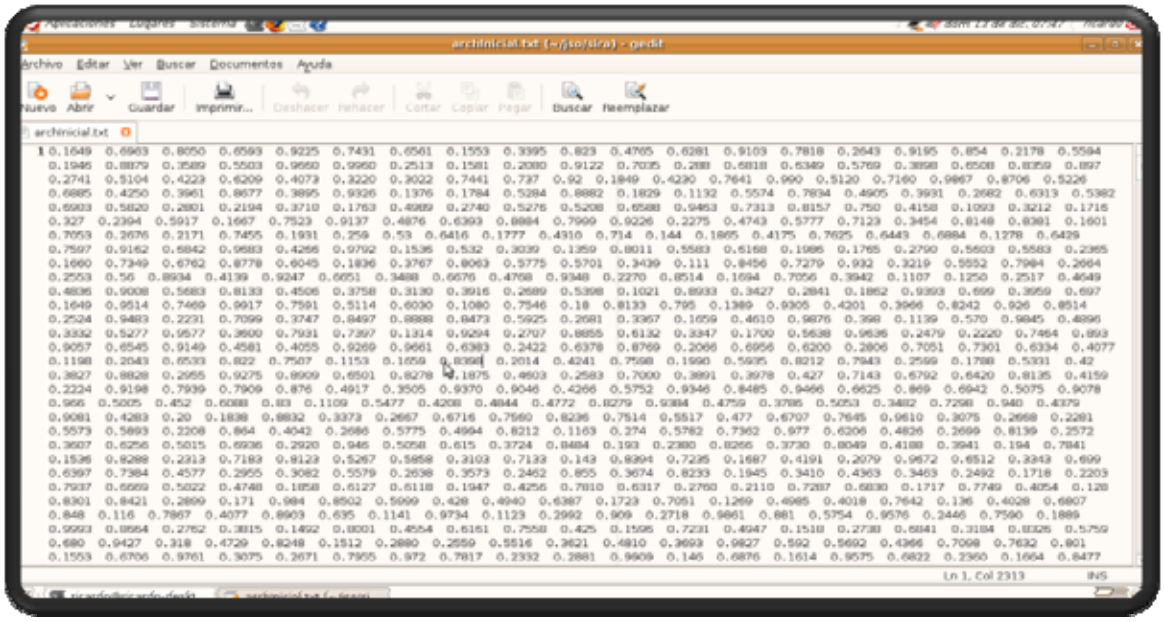

Figura 5: Contenido del archivo de entrada archInicial.txt

Para realizar las pruebas se ejecutó 5 veces el programa con el mismo número de procesos variando el tiempo de ejecución en milésimas de segundo, pero obviamente puede variar el tiempo hasta en segundos si se le asigna alguna tarea ajena a MPI, las pruebas correspondientes se realizaron dedicándole todos los recursos de Hardware disponibles a MPI.

Los resultados como se menciono anteriormente son de tipo flotante y tendrán un rango de (0,1) y se muestran en pantalla para que el usuario final los pueda interpretar y pueda determinar si la estructura se encuentra dañada o no.

Durante la ejecución cada proceso creado se apodera de un núcleo, es decir durante las pruebas stand-alone cuando se creaban 3 subprocesos (np) solo trabajaban 3 núcleos al 100% quedando un núcleo disponible, y cuando se hizo la prueba con 5 subprocesos primero empezaban 4 procesos es decir se utilizaban los cuatro núcleos del CPU y el primer subproceso que acababa le cedía sus recursos de hardware al que faltara (Véase Figura 6).

Cuando se realizaron la pruebas con ejecución remota sucedía lo mismo, por ejemplo cuando se realizó la prueba con 6 subprocesos la primer máquina que fungía como servidor llevaba a cabo 3 de los 6 subprocesos y enviaba a ejecutar remotamente los otros 3 subprocesos faltantes, si el servidor llegara a acabar primero espera la respuesta del nodo que ejecutó remotamente.

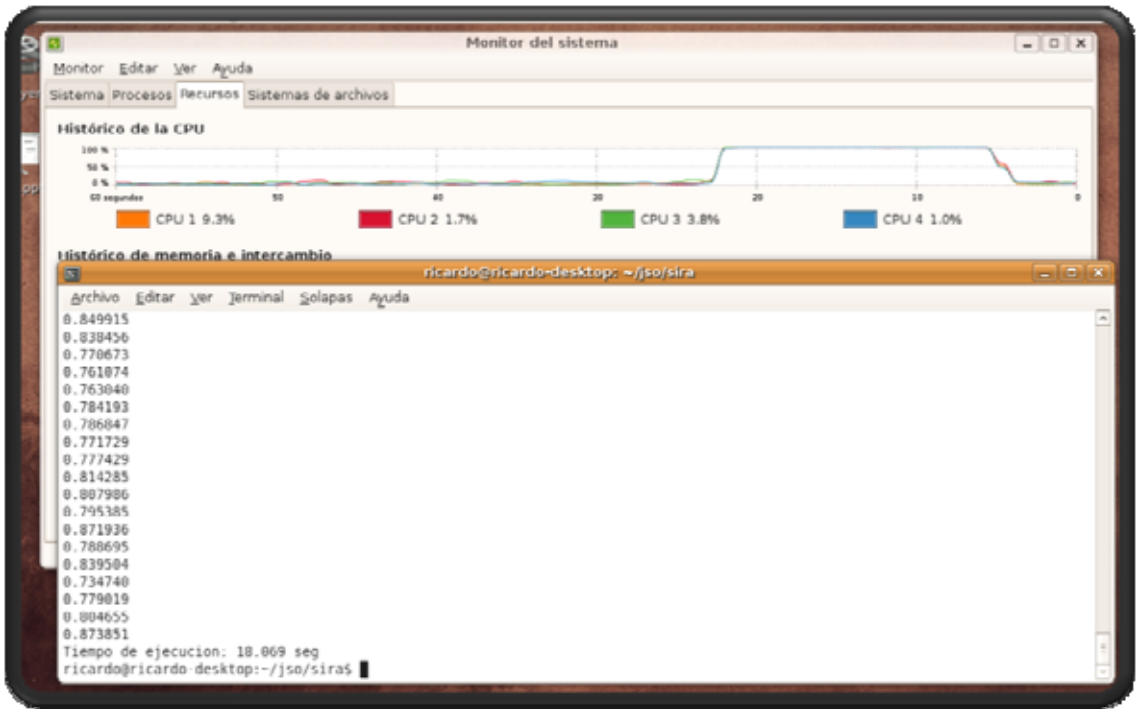

Figura 6: Se muestra los datos de salida que el programa despliega en pantalla cuando finaliza así como el tiempo de ejecución, también se muestra el trabajo de los 4 núcleos durante la ejecución.

Para las pruebas se utilizaron dos CPU con Procesador Intel Core 2 Quad 2.4GHz y 4Gb en RAM cada una, es decir se simulaban 8 CPU.

Los resultados obtenidos fueron los siguientes:

Número de Procesos	Tiempo(1 CPU Quad Core)	Tiempo(Ejecución Remota)
4	5.021	4.454
5	7.008	4.260
6	7.569	4.057
7	8.352	4.208
8	9.710	4.889
9	11.013	5.690
10	11.946	5.905
11	13.012	6.914
12	14.342	7.349
13	15.334	8.544
14	16.847	8.985

A continuación se mostrara gráficamente los resultados desde diferentes interpretaciones (Véanse Figuras 7,8 y 9).

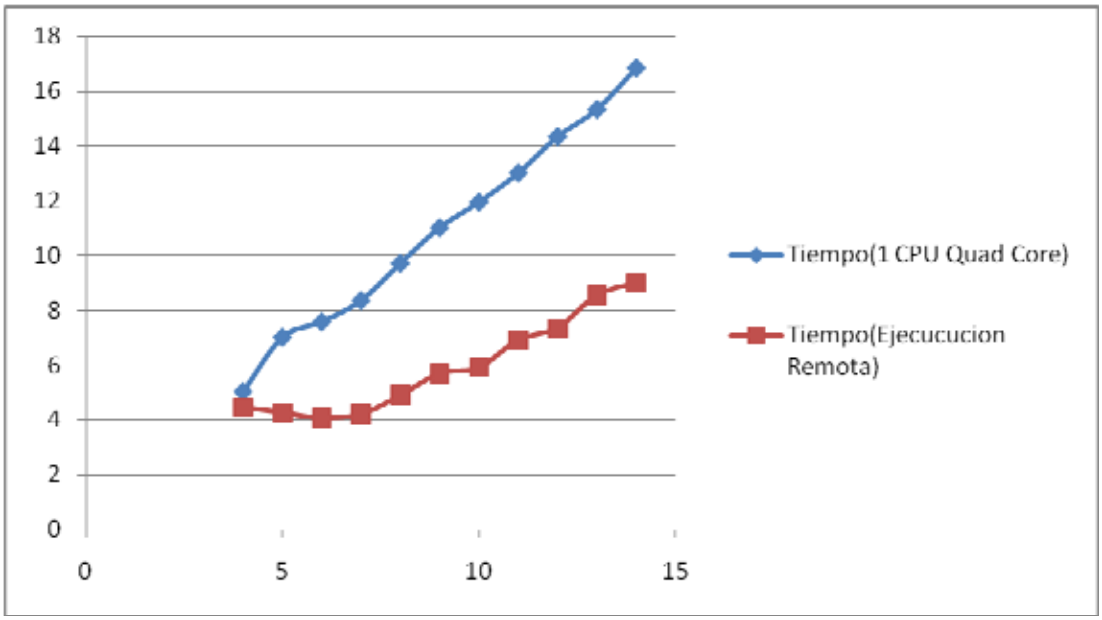

Figura 7: Muestra claramente el desempeño teniendo 4 y 8 núcleos.

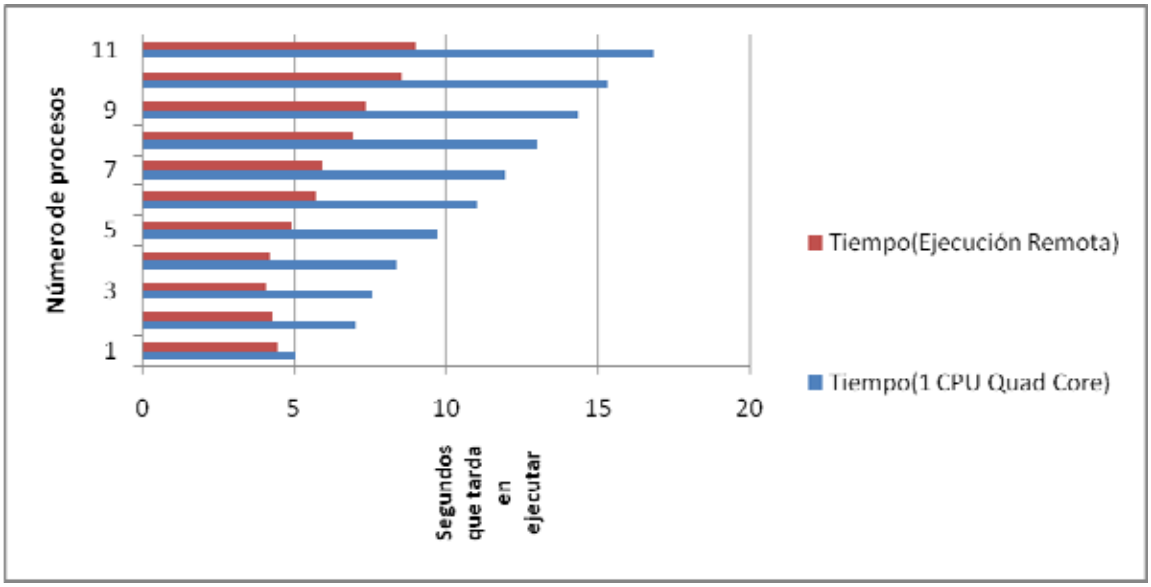

Figura 8: Muestra claramente el desempeño teniendo 4 y 8 núcleos y el número de procesos.

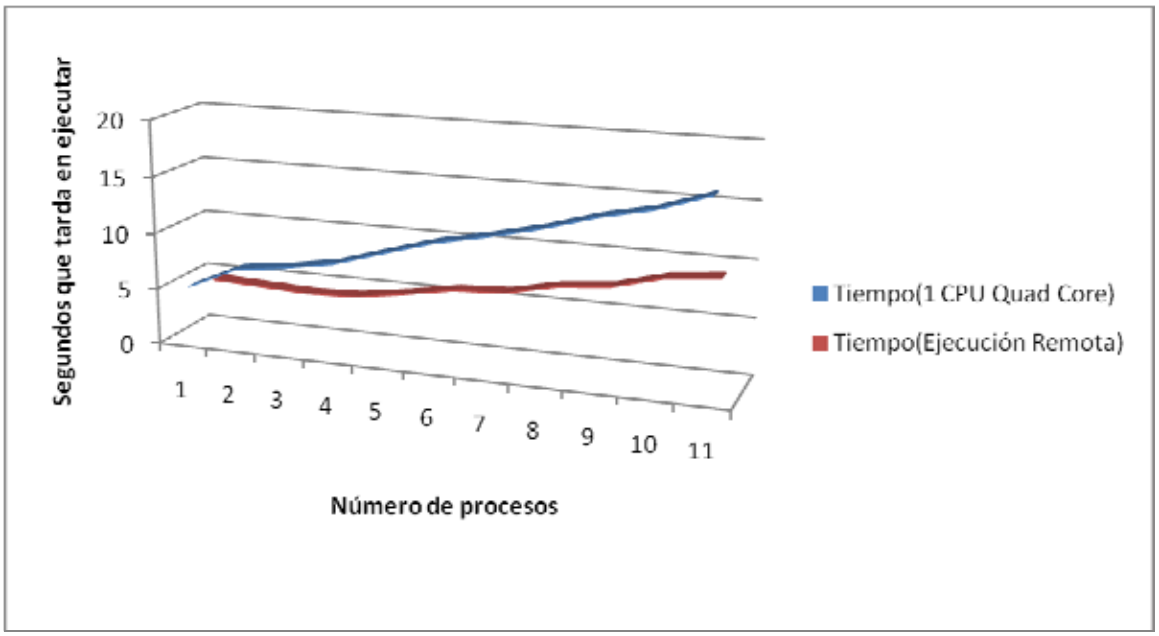

Figura 9: Muestra claramente el desempeño teniendo 4 y 8 núcleos y el número de procesos

Con estos resultados se puede observar que se pudo disminuir casi un 50% del tiempo de ejecución utilizando 8 núcleos.

5. TREX

La herramienta de Ejecución Remota Transparente TREx (Transparent Remote Execution) fue diseñada para aprovechar las estaciones de cómputo que comparten una red y que por lo regular no explotan sus recursos (memoria RAM y ciclos de CPU en particular).

Con TREx se pueden explotar estas estaciones a fin de que puedan ser utilizados los ciclos de CPU ociosos y realizar diversas tareas que requieran grandes cantidades de recursos de forma transparente al usuario principal.

5.1 CARACTERISTICAS DE TREx

Los componentes básicos dentro de TREX son, maestro demonio, que se ejecuta en estaciones de trabajo con gran carga de trabajo es decir el nodo maestro que delegara trabajo entre los esclavos, y el esclavo demonio, que se ejecutan en estaciones con baja carga de trabajo.

Las estaciones maestras pueden delegar trabajo a todas las estaciones esclavas registradas. El nodo Maestro(s), se encargara de enviar el ejecutable (servicio), y, en determinado caso los archivos de entrada, a sus esclavos. Los esclavos pueden ejecutar remotamente cualquier servicio que el maestro envía y regresar la salida al maestro como si el programa se ejecutara a nivel local. Toda ejecución se hace transparente es decir el usuario de la estación no tiene conocimiento del proceso a distancia que está ocurriendo. Un maestro también puede ser un esclavo y puede procesar su propio trabajo que se delega a sí mismo o procesar trabajo recibido de otros maestros.

En la *figura 10* se muestra el funcionamiento del algoritmo con MPI

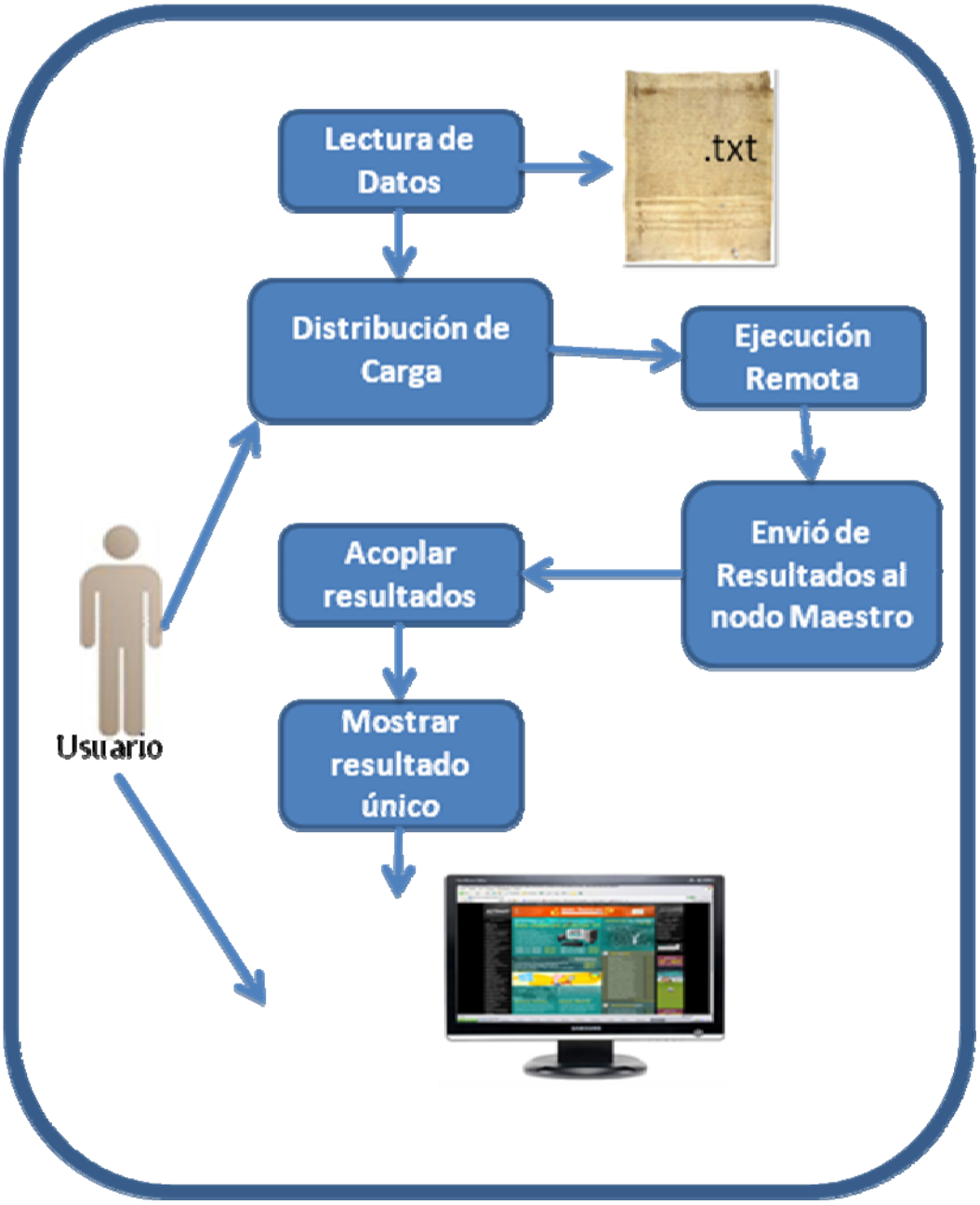

Figura 10: Diagrama a bloques correspondiente al funcionamiento de TREx con el algoritmo A basado en MAC, donde el usuario interviene en decidir a qué nodos se distribuirá la carga.

1. MAESTRO

La función principal del maestro demonio es delegar el trabajo a los esclavos y recibir la producción que estos generen como puede ser un archivo de salida.
El maestro debe tener un archivo script.mst que contiene la información de servicios incluyendo los recursos necesarios para todos los servicios que serán ejecutados.

El archivo script.mst debe contener el siguiente formato:

<SERVICE> /* Apertura de servicio*/
<NAME></NAME> /* Nombre del archivo, de preferencia un número que lleve el consecutivo de los servicios*/
<SERVICENAME> </SERVICENAME> /*Es el nombre de la carpeta donde se encuentran los servicios a ejecutar incluyendo los archivos de entrada. */
<EXECUTABLE></EXECUTABLE> /* Es el nombre del archivo ejecutable incluyendo su extensión */
<PARAMETERS></PARAMETERS> /*Los parámetros que se pasaran al ejecutable */
<DIRECTORY> </DIRECTORY> /* Directorio donde se colocara el archivo de respuesta por parte de los esclavos */
<DEPENDENCY></DEPENDENCY> /* Dependencia que hay entre servicios, es decir un servicio de nombre “dos” puede necesitar que se ejecute primero servicio “uno” para poder utilizar su archivo de salida. */
</SERVICE> /* Cierre del servicio */

El maestro analiza este archivo, obtiene toda la información necesaria sobre los servicios que serán enviados, e invoca a todos los esclavos disponibles dentro de su grupo para la ejecución remota.
Todos los servicios (ejecutables, archivos de entrada, bibliotecas, etc.) deben estar presentes en el maestro.

El maestro también debe utilizar un archivo IP.cfg. Este archivo contiene información acerca de todos los esclavos que están registrados en el maestro y obviamente que pueden ser utilizados para brindar servicios. Periódicamente, el maestro envía mensajes de registro a todas las direcciones IP que se encuentran en el archivo IP.cfg. Si una dirección IP no responde en un determinado número de intentos de registro, la comunicación, se presume muerta y todos los servicios que se ejecutan en esta ubicación son reprogramados.

El archivo IP.cfg debe tener el siguiente formato:

10.193.129.111
10.193.130.223
10.193.130.222
10.167.129.211
.
Todos los esclavos deben tener un único IP.cfg dentro de un archivo con el fin de evitar que maestros codiciosos usen los recursos de todos los esclavos en el sistema.

Cuando se produce un error en la ejecución remota se informa al maestro, quien debe reprogramar el servicio con otra estación de trabajo. La reprogramación ocurre cuando se bloquea un esclavo, no tiene ya los recursos disponibles,

devuelve un código de salida negativo que indique un problema con la ejecución, o la comunicación entre maestro y esclavo muere.

2. ESCLAVO

Las funciones principales del esclavo demonio es aceptar trabajo de los nodos maestro, aceptar cualquier servicio de archivos si es necesario, ejecutar el volumen de trabajo que se le encomienden, y enviar la salida al maestro.

El esclavo también debe comprobar constantemente si sus recursos no entran dentro de un determinado umbral, es decir que los servicios que va a proporcionar no excedan o saturen sus recursos disponibles, si esto sucede, entonces, el esclavo terminará la ejecución remota, borrara el directorio donde tuvo lugar la ejecución remota, y enviara un informe de error de vuelta al maestro.

5.2 COMUNICACIÓN EN TREx

La comunicación entre un maestro y esclavo tiene varios pasos.

El primer paso es que el esclavo se registre con un maestro. El maestro enviará mensajes de solicitud de registro a todas las direcciones IP encontradas en su archivo IP.cfg en intervalos constantes. Cuando un esclavo recibe un mensaje de solicitud de registro, se enviará un mensaje al maestro con su RAM disponible, espacio de disco duro disponible, y la utilización del procesador es entonces cuando el maestro decidirá si el esclavo está "inactivo" basándose en la información sobre los recursos del esclavo. Sólo los esclavos inactivos serán registrados al maestro y todos los demás esclavos se ignorarán hasta que el próximo mensaje de registro se reciba con actualizaciones de los valores de los recursos.

Una vez que el maestro haya delegado sus servicios, se comprueba si los esclavos tienen los archivos necesarios para su ejecución. Si no los tienen, entonces el maestro enviará lo que falta. Después de que el maestro envía todos los archivos necesarios para el esclavo, debe enviar el comando de ejecución.

Cuando el esclavo recibe el comando de ejecución, ésta se inicia. Después de que la ejecución se ha completado, el esclavo envía los archivos de salida al maestro a un directorio pre-definido. El esclavo envía el código de salida, es decir, un valor numérico que indica si la ejecución tuvo éxito o no. Cuando la ejecución es satisfactoria, el maestro puede continuar la designación de cualquier servicio a sus esclavos.

5.3 PASOS PARA EL ÓPTIMO FUNCIONAMIENTO DE TREx

1. El nodo maestro lanzara un demonio a todos los nodos esclavos que estén registrados en su archivo IP.cfg, esto lo hará con la ayuda de la interfaz grafica que proporciona TREx Master
   Inicialmente se encuentra como lo muestra la figura, donde no muestra ningún esclavo disponible por el momento.
   Simplemente después de haber ingresado las IP's dentro del IP.cfg se da un "click" en el botón "Look for Slaves".

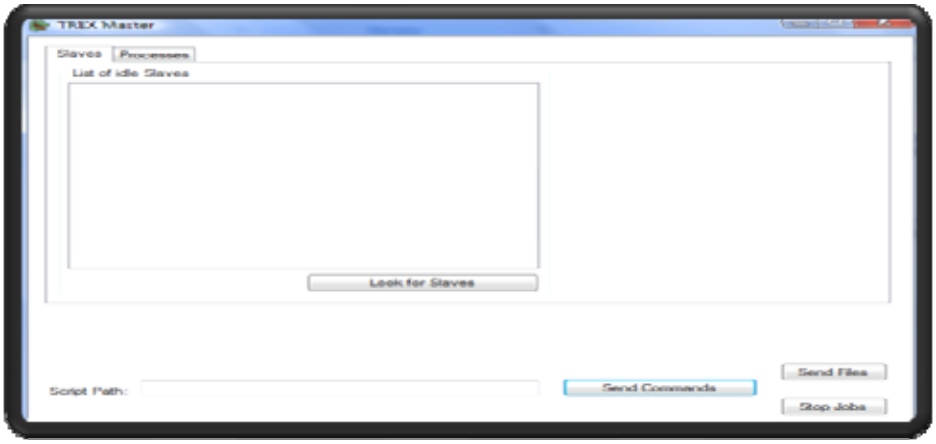

Figura 11: Interfaz de TREx Master

Aparecerán todos los esclavos disponibles y se ingresa la dirección donde se va a localizar el archivo script.mst

2. Se instalaran todos los servicios a ejecutar asi como los archivos necesarios de entrada en la carpeta creada por default durante la instalacion de TREx Master, esta carpeta se lleba el nombre de Services.(Vease figura 12)

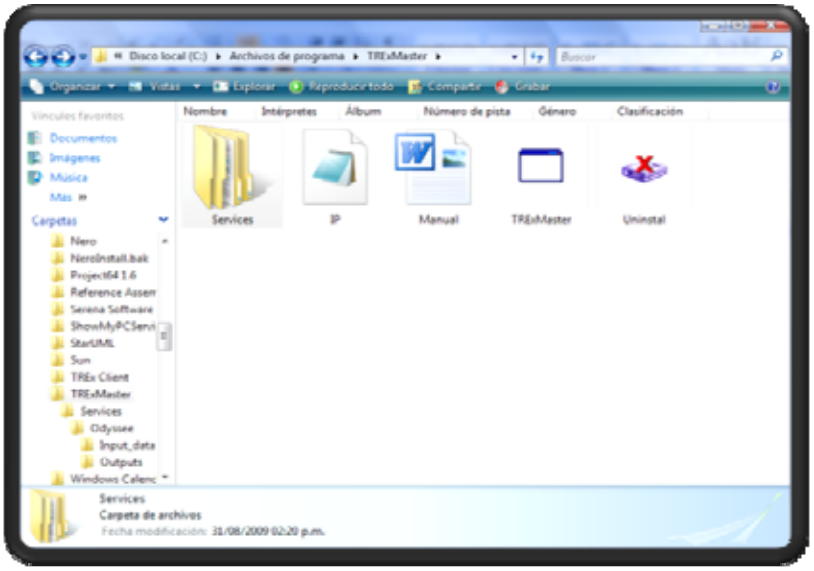

Figura 12. Muestra el contenido de la carpeta que se crea al instalar TREx Master, y se observa la carpeta Services donde se instalaran todos los servicios a ejecutar

3. Todos los nodos esclavos que tengan instalado el TREx Client, le responderán al nodo maestro si es que están o no disponibles para realizar alguna tarea, así como también en la interfaz se podrá observar si los esclavos cuentan con los archivos necesarios para la ejecución de algún servicio, de no ser así se pueden enviar todos los archivos necesarios seleccionando la IP del esclavo y presionando el botón de Send Files, así el nodo maestro enviara todos los archivos contenidos en la capeta Services.

   El usuario que controla TREx Master se podrá dar cuenta cuando el esclavo recibió satisfactoriamente todos los archivos ya que los marca con una insignia de correcto cuando se enviaron satisfactoriamente. (Véase Figura 13)

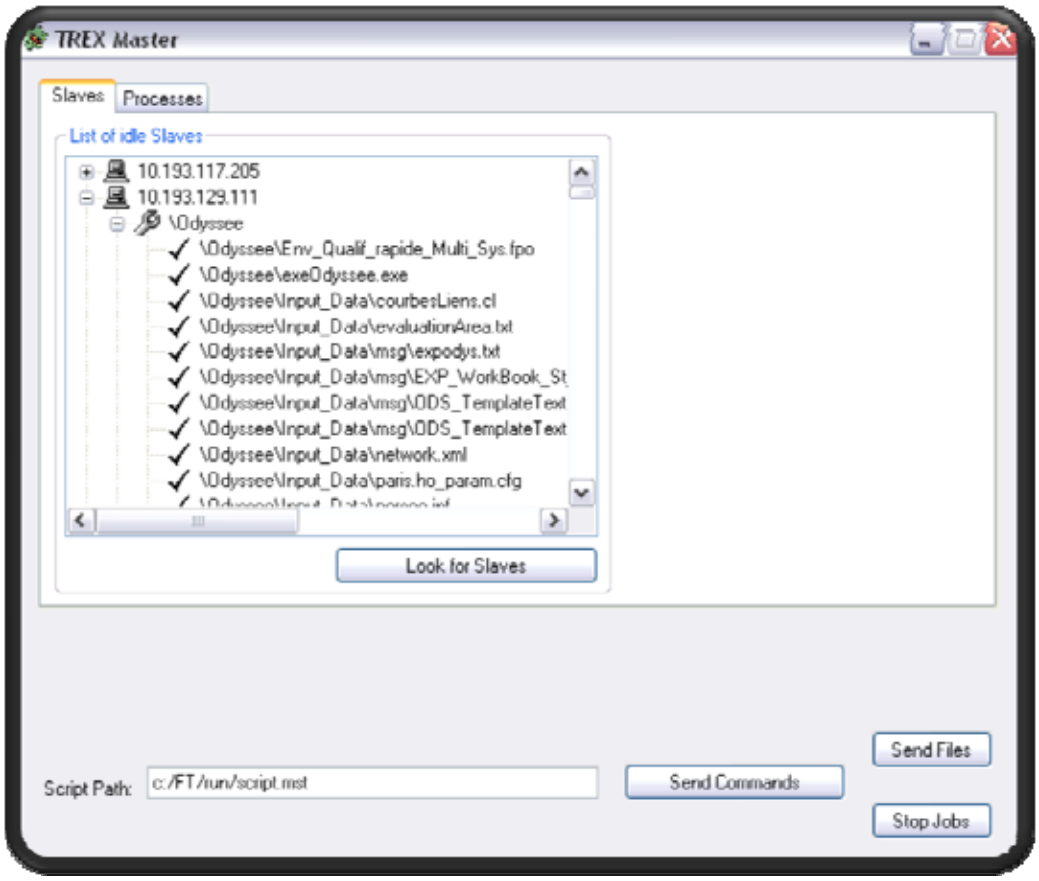

Figura 13. Muestra que uno de los dos esclavos recibió todos los archivos necesarios para la ejecución remota satisfactoriamente.

4. Se configura el archivo script.mst con cada uno de los atributos explicados anteriormente.

5. Una vez enviados los archivos y teniendo el script, se presionara el botón Send Commands, el cual iniciara la ejecución en el orden descrito por el srcript.

   TREx Master tiene la capacidad de seleccionar automáticamente y de manera dinámica aquellos nodos esclavos que tengan más recursos disponibles (hardware), así que al nodo que se encuentre disponible se le dará la tarea de ejecutar lo que el usuario desee.

   Trex Master dará un informe acerca del estado en que se encuentran los programas a ejecutar, es decir, si se encuentra en “envío”, “ejecución” o ya finalizó el proceso. (Véase Figura 14)

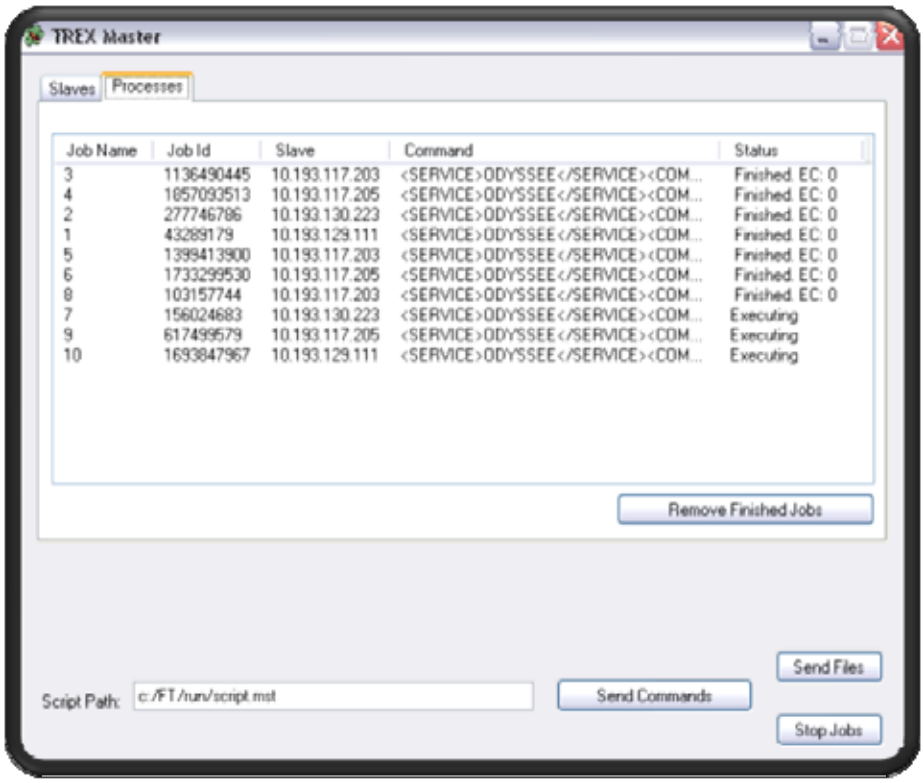

Figura 14. Muestra el estatus de cada uno de los servicios, así como la IP de la PC donde se está ejecutando.

6. Finalmente los resultados de la ejecución remota serán recibidos en alguna carpeta elegida por el usuario desde el script.

El archivo lleva el nombre console_output.

5.4 PRUEBAS REALIZADAS CON TREx

Para las pruebas que se realizaron se usaron 3 Computadoras, dos de ellas de Procesador Intel Core 2 Quad con 4Gb en RAM cada una.
La otra computadora fue una Pentium 4 con 1Gb en RAM y la topología que se uso para este caso fue la siguiente (Véase Figura 15):

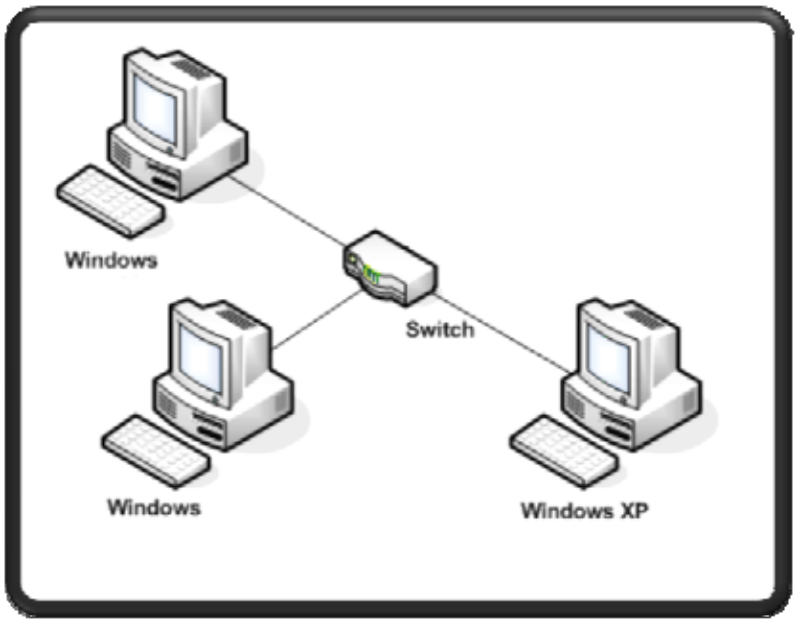

Figura 15: Topología usada en experimento

Se puede elegir cualquier topología ya que TREx así lo permite, de hecho dentro del archivo IP.cfg pueden existir direcciones IP de diferente segmento, es decir que pertenezcan a otra LAN, obviamente es necesario que exista la conexión necesaria entre routers, switch y cualquier otro dispositivo que permita que el servidor tenga a su alcance todas las registradas en el archivo antes mencionado.

De la misma manera que con MPI se uso el mismo programa “MAC”, para poder observar el desempeño de TREx se comparara con los tiempos que se obtuvieron con MPI.

Los resultados de TREx son los siguientes donde también se muestra el tiempo hecho por MPI para poder hacer comparaciones.

Número Procesos	MPI	TREx
4	4.454	8.0
5	4.260	8.0
6	4.057	9.7
7	4.208	12.5
8	4.889	13
9	5.690	15.7
10	5.905	18
11	6.914	20
12	7.349	21
13	8.544	23
14	8.985	25

Nota: Para Trex se utilizó una máquina más pero de menos poder que las usadas con MPI además de que TREx cada vez que ejecuta envía los archivos necesarios (archivos de entrada) y todo ese tiempo lo toma como tiempo de ejecución, en cierta manera puede ser una desventaja de TREx en cuestión de tiempo, pero en MPI cada vez que se le hace una modificación al código fuente o a los archivos de entrada es necesario actualizarlo en todos los nodos lo cual puede ser más costoso en ese momento para MPI que para TREx, ya que como se mencionó TREx puede enviar los archivos de entrada como el ejecutable cada vez que se pide un servicio remoto. (Véanse Figuras 16 y17)

Una vez terminada la ejecución, los resultados se guardan en el archivo que ya se menciono en la carpeta elegida por el usuario, el archivo se llama console_output arrojando datos esperados como estos:

0.805405
0.796972
0.738452
0.865755
0.785790
0.749238
0.813949
0.741079
0.820097
0.777701
0.763083

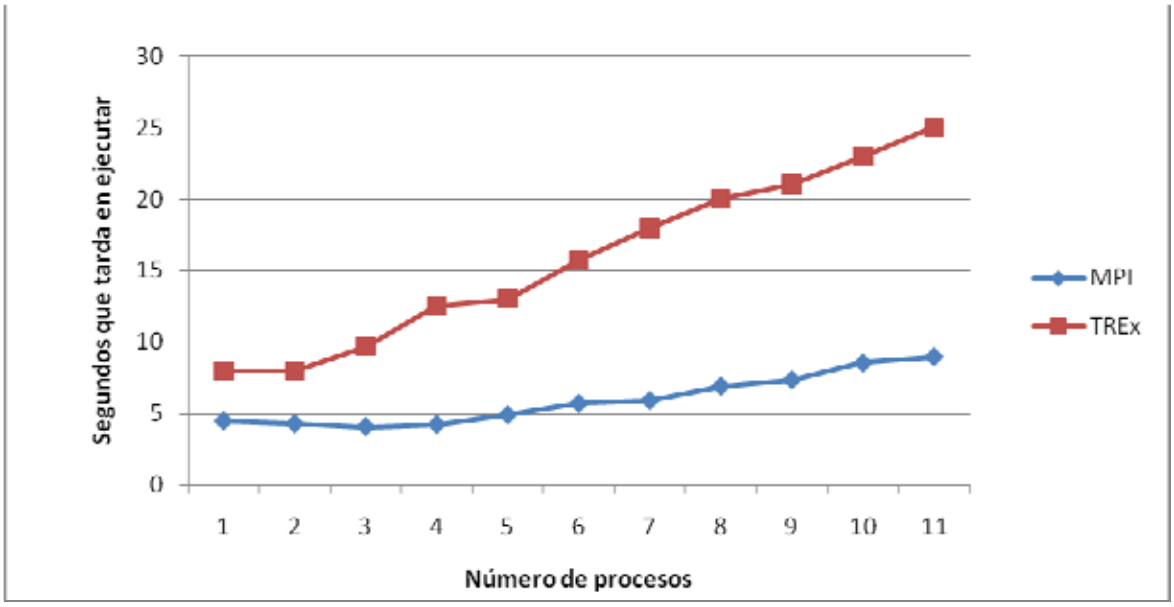

Figura 16. Muestra la comparación números de procesos contra tiempo, donde podemos observar que MPI fue muy superior en este caso.

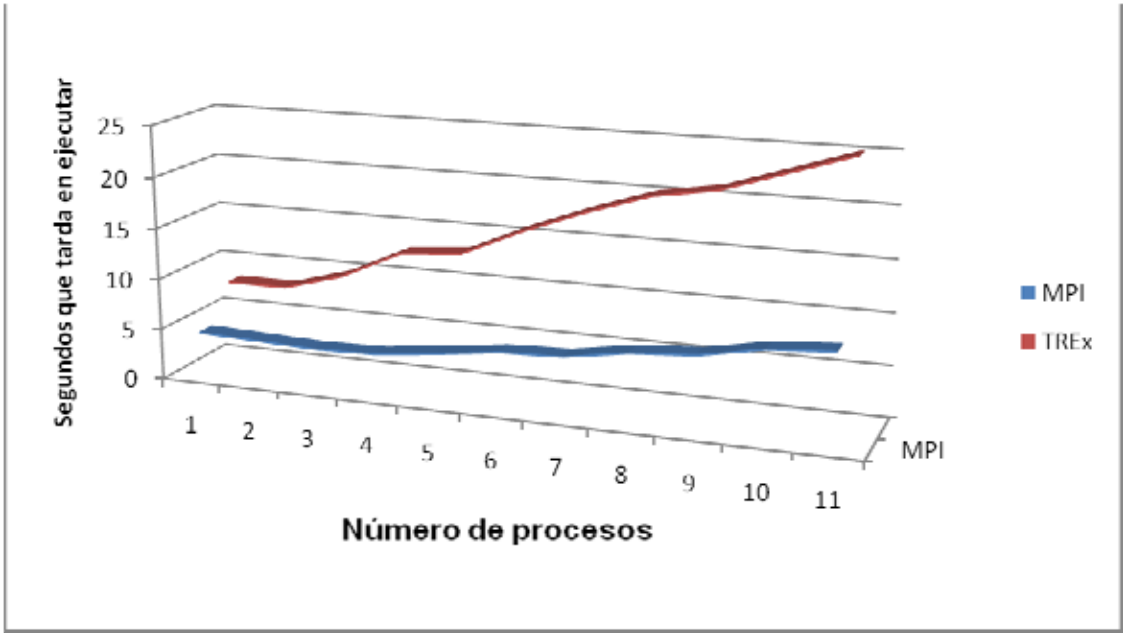

Figura 17. Muestra en diferente perspectiva la comparación de números de procesos contra tiempo, donde podemos observar que MPI fue muy superior en este caso.

6. PRUEBAS ADICIONALES:

Se realizaron pruebas con programas que proponen los desarrolladores de MPI en su página electrónica oficial que trata de arrojar como salida un número cada vez más exacto de lo que es π=3.1416.

Este programa está basado en un algoritmo que usa una aproximación rectangular a la integral de la función f(x) = 4/(1+x*x).

En este caso este algoritmo se adaptara para hacer las pruebas correspondientes con TREx.

En el siguiente cuadro se describirá el resultado como los tiempos de ejecución tanto en TREx como en MPI.

Numero de Procesos	MPI	TREx
2	**12.416**	**9**
4	**6.978**	**13**
5	**7.724**	**13**
6	**7.168**	**15**
7	**6.956**	**15**
8	**6.653**	**16**
9	**6.845**	**16**
10	**7.985**	**18**

Los resultados en MPI se muestran en la figura 18.

Los resultados obtenidos en TREx son los que se muestran a continuación:

Valor de pi: 1.570796
Valor de pi: 0.758398
Valor de pi: 0.628959
Valor de pi: 0.523599
Valor de pi: 0.448799
Valor de pi: 0.392699
Valor de pi: 0.349066
Valor de pi: 0.314159

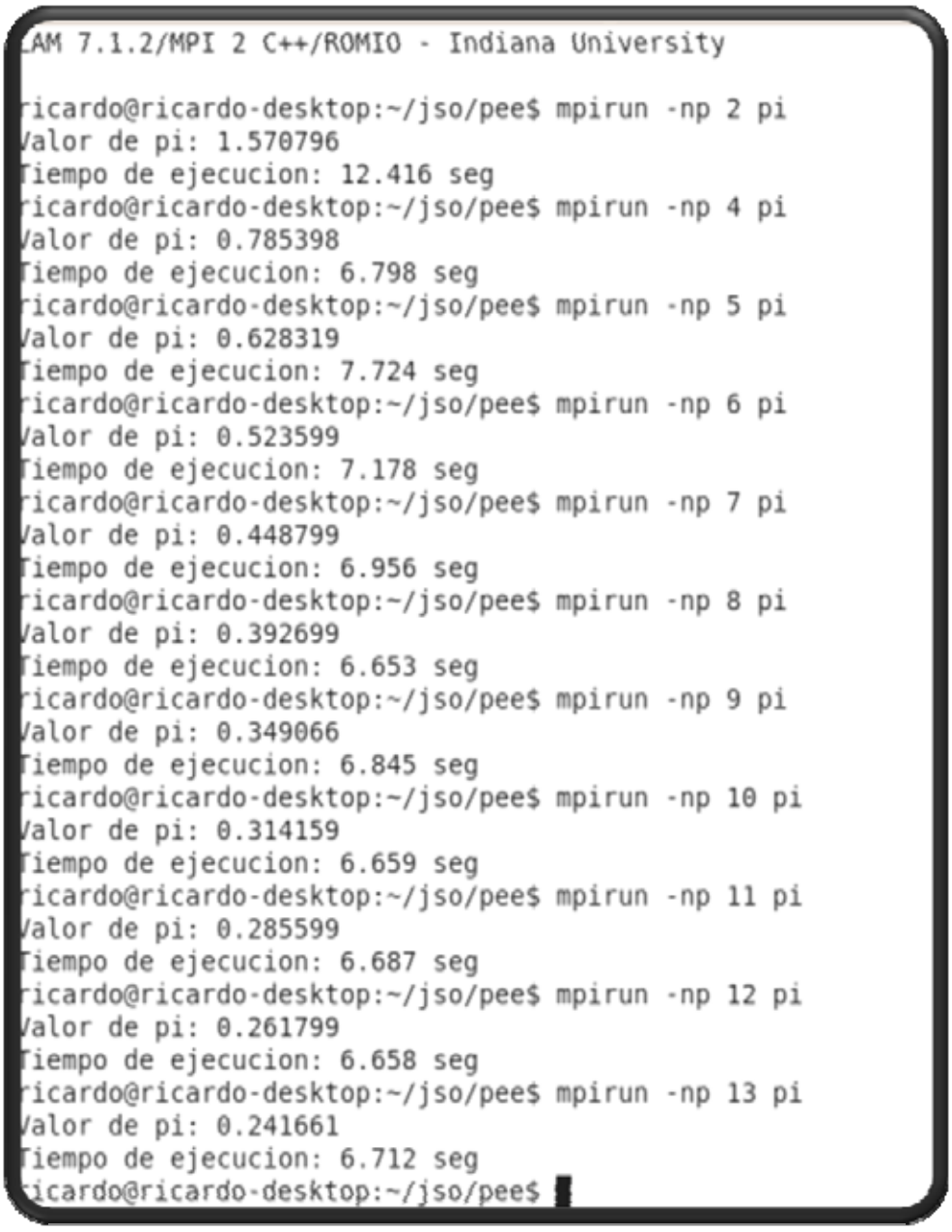

Figura 18: Muestra varias ejecuciones del programa pi.c desde el sistema Linux

7. CONCLUSIONES

Puedo decir que para los usuarios de alguna distribución de Linux, MPI es una gran herramienta para paralelizar algún programa, además de que MPI tiene mucho más tiempo de vida en comparación con la de TREx, es decir me parece un poco más confiable y seguro además que en las pruebas que realice resultó a mi punto de vista ser más eficiente MPI que TREx.

Ahora hablando un poco del algoritmo que implementamos me parece que es de gran ayuda ahora que en México y existen construcciones muy importantes como lo son puentes y edificios, por ejemplo el Golden Gate que es un famoso puente situado en California, Estados Unidos es monitoreado por un algoritmo similar a MAC ya que tiene conectados miles de sensores en toda su estructura y todas señales son registradas en computadoras que como ya mencione a través de un algoritmo similar a MAC detectan daños en la estructura.

La implementación del algoritmo de MAC con las librerías y funciones de MPI mostró una disminución importante en el tiempo de ejecución con respecto a la versión secuencial.

Como lo mencione para MPI se utilizó una distribución Linux Ubuntu 8.10 con la cual me encontré con varios conflictos, el más importante desde mi punto de vista es la inestabilidad de las distribuciones Linux, ya que durante las instalaciones de los paquetes necesarios para la compilación y el buen funcionamiento de MPI se pueden producir varios errores, el más común para mí fue instalar los repositorios de LAM, y cuando empezaba a bajar del internet algunos de los paquetes no los instalaba correctamente y me fue necesario reinstalar continuamente Ubuntu debido a estos errores durante la instalación de bibliotecas y rutinas de MPI.

Desde mi punto de vista una de las distribuciones más fiables para este tipo de trabajo rudo es la distribución de Linux Debian, lamentablemente su estabilidad no es proporcional con su manejo ya que puede ser tediosa la configuración de un cluster con MPI en esta distribución por el número de pasos que se deben realizar, y uno de mis objetivos es mostrar que es una herramienta completa y de fácil uso.

MPI a pesar de ser una herramienta poderosa no es muy usada o no se le podríamos dar el mismo uso como a TREx. Por ejemplo, si nos encontramos en una oficina donde queramos usar los recursos de las máquinas que no están operando al 100% sería un poco difícil, ya que en primer lugar necesitamos instalada una distribución de Linux y en muy pocas empresas manejan estos sistemas por la misma inestabilidad que presentan o simplemente por desconocimiento del Software Libre, es decir MPI es usado en la mayoría de las veces en lugares donde se maneje el super-computo, una prueba de esto es que MPI ha sido de gran influencia para trabajos desarrollados por empresas como son IBM e INTEL entre otras.

TREx es una herramienta similar a MPI que nos sirve para la ejecución remota, solo que está hecha para distribuciones de Windows es decir, primero debemos contar con un licencia de Windows para poderlo instalar.

Después de la instalación de TREx encontré algunos conflictos ya que inicialmente instale TREx Master en Windows 7 y cuando ejecutaba TREx Master tenía conflictos con la comunicación, ya que el Firewall de Windows no lo dejaba trabajar satisfactoriamente además de que existía incompatibilidad entre Windows y TREx Master.

Pero puedo decir que TREx trabaja de excelente manera con Windows Vista en todas sus versiones al igual que Windows XP, el único conflicto es que tenemos que deshabilitar los Firewall tanto de los nodos Clientes como de los nodos Maestro.

Trex me parece un muy buena herramienta de muy fácil uso, de hecho gracias a sus interfaces es más sencillo la instalación, configuración y manejo. Trex podría ser usado en casi cualquier sitio que requieran ejecución remotamente y tengan alguna Licencia de Windows, pero creo que a diferencia de MPI no es tan eficiente y por lo tanto lo considero viable para la ejecución de pequeñas tareas o simplemente si no se tiene conocimiento de la programación paralela esta es una muy buena herramienta que evitara de problemas a muchos.

Para finalizar puedo decir que tanto TREx y MPI son herramientas poderosas pero tal vez sus ventajas y desventajas de cada uno no sean propias de los sistemas, si no de las limitaciones de cada uno de los sistemas operativos donde se tiene que instalar cada una de estas herramientas.

Por lo tanto la demanda de mayor poder de procesamiento por parte de los usuarios de aplicaciones de cómputo intensivo, así como la necesidad de lograr un mejor aprovechamiento de los recursos, ha llevado a una creciente proliferación en el uso de clusters. Frecuentemente, debido a la falta de herramientas apropiadas para todo tipo de aplicaciones y configuraciones de clusters, es necesario que los usuarios tengan que desarrollar por sí mismos sus propias herramientas y aplicaciones. El presente trabajo es una contribución para todas aquellas personas que quieran iniciarse en el mundo de los clusters

8. REFERENCIAS

- Wang, Richert, Cauich, Enrique, Sherson, Isaac D.,
  Federated Clusters Using the Transparent Remote Execution (TREx)
  Julio 2007

- Aitor, Viana, Hombrados, Jesús, López, Juan F.
  Sistemas LAM/MPI
  Septiembre 2008

- Carpio, Pacheco, César, Ruiz, Sandoval, Hernández, Manuel,.
  Sociedad Mexicana de Ingeniería Estructural
  Identificación de Daño Estructural en un Marco Plano a Cortante
  Abril 2008

- http://www.mcs.anl.gov/research/projects/mpi/
  Septiembre 2009

- http://www.lam-mpi.org/
  Abril 2009

- http://telematica.cicese.mx/computo/super/calafia/programacion/mpi.php
  Septiembre 2009

- http://www.mcs.anl.gov/research/projects/mpich2/downloads/index.php?s=downloads
  Julio 2009