Resolución de Cadenas de Markov con *Clusters* de PCs de Alto Rendimiento

Mejía Raúl D.
Fernandez Diego A.
Bernal Iván M.

Escuela Politécnica Nacional

***Resumen*—Se presenta una visión general de computación de alto rendimiento utilizando clusters de computadoras personales (PCs). Luego se presentan aspectos relacionados al diseño e implementación de un cluster prototipo que utiliza Linux y dos toolkits de instalación automática (OSCAR y Rocks). A continuación, se describe el desarrollo de una aplicación para resolver Cadenas de Markov a Tiempo Continuo (CMTC) y Cadenas de Markov a Tiempo Discreto (CMTD) utilizando la librería de paso de mensajes MPI. Finalmente, se exponen los resultados comparativos obtenidos al ejecutar las aplicaciones desarrolladas en el cluster prototipo (variando el número de nodos) y en una sola máquina.**

***Términos para indexación*— Cadenas de Markov, Cluster, MPI.**

I. INTRODUCCIÓN

La computación paralela surge como una repuesta para suplir los requerimientos y necesidades de procesamiento de muchas aplicaciones y sistemas, que además demandan obtener sus resultados de forma óptima y eficiente.

Entre las arquitecturas prominentes que se han presentado como soluciones se pueden citar: Procesadores Masivamente Paralelos (MPP–Massively Parallel Processors) que provee una arquitectura en la que no se comparten sus dispositivos o el sistema operativo [1]; los Multiprocesadores Simétricos (SMP–Symmetric Multiprocessors) en la que se comparten menor

ia y dispositivos de entrada/salida entre los distintos procesadores, pero que escalan adecuadamente solo hasta alrededor de 128 CPUs (Central Processing Units) [1], [2]; los sistemas de Acceso a Memoria No Uniforme con Coherencia de Caché (CC-NUMA Cache-Coherent Non-Uniform Memory Access) [3]; los Sistemas Distribuidos que son redes convencionales de computadoras independientes; los clusters [2], [4]; y los grids que están estructurados por un conjunto de recursos computacionales heterogéneos, distribuidos y pertenecientes a diferentes organizaciones [4].

Este trabajo se realizó gracias al financiamiento de La Escuela Politécnica Nacional (EPN) y de la Fundación para la Ciencia y la Tecnología (FUNDACYT), en el Departamento de Electrónica, Telecomunicaciones, y Redes de Información de la EPN.

D. R. Mejía participó en el proyecto de clusters por la Escuela Politécnica Nacional (e-mail: david_dam33@hotmail.com).

D. A. Fernández participó en el proyecto de clusters por la Escuela Politécnica Nacional (e-mail: dfernandez@utpl.edu.ec).

I. M. Bernal trabaja en la Escuela Politécnica Nacional en el Departamento de Electrónica, Telecomunicaciones, y Redes de Información, Ladrón de Guevara E11-253, Quito-Ecuador (teléfono: 5932-2507-144; fax: 5932-2547-175; e-mail: imbernal@mailfie.epn.edu.ec).

II. CLUSTERS: VISIÓN GENERAL

Un cluster es una solución computacional conformada por un conjunto de sistemas computacionales muy similares entre si, interconectados mediante alguna tecnología de red de alta velocidad, configurados de forma coordinada para dar la ilusión de un único recurso; cada sistema estará proveyendo un mismo servicio o ejecutando una (o parte de una) misma aplicación paralela. La característica inherente de un cluster es la compartición de recursos: ciclos de CPU (Central Processing Unit), memoria, datos y servicios.

En el año de 1994 se dispuso del primer cluster conocido como Beowulf, cuyo trabajo se debe al grupo de investigación conformado por T. Sterling y D. Becker en CESDIS (Center of Excellence in Space Data and Information Sciences) y auspiciado por el Proyecto de la Tierra y Ciencias del Espacio (ESS). Beowulf estuvo basado en componentes genéricos COTS (Components Of The Shelve), cuyo rendimiento era de 3.2 GFLOPS. Debido a las deficiencias de los Sistemas SMPs, la disponibilidad de herramientas estándar para computación distribuida (MPI-Message Passing Interface [5], PVM-Parallel Virtual Machine [6]), y gracias a la difusión de Beowulf a través de la NASA y comunidades académicas, los clusters se convirtieron rápidamente en una alternativa efectiva de procesamiento paralelo para satisfacer requisitos de cómputo especificos.

A. Clusters: Clasificación

Los clusters se pueden clasificar en base a los usos y servicios que ofrecen: alto rendimiento, alta disponibilidad y alta eficiencia [7].

El primero conocido como de alto rendimiento, para tareas que requieren de gran capacidad de cómputo (procesamiento, memoria o ambos) y tareas que podrían requerir la utilización de los recursos por largos periodos de tiempo. El segundo denominado de alta disponibilidad, el cual brinda máxima disponibilidad de los servicios que se ofrecen, además de un rendimiento sostenido. El último de alta eficiencia, cuyo

objetivo primordial es culminar el mayor número de tareas en el tiempo más corto posible.

Los clusters se los puede también clasificar como clusters de IT comerciales (alta disponibilidad, alta eficiencia) y clusters científicos (alto rendimiento) [8].

A pesar de las discrepancias a nivel de requerimientos de las aplicaciones, muchas de las características de las arquitecturas de hardware y software, que están por debajo de las aplicaciones en todos estos clusters, son las mismas. Más aún, un cluster de determinado tipo, puede también presentar características de los otros.

B. *Clusters: diseño, instalación y configuración*

Existen ciertos tópicos que son importantes el momento de realizar el diseño, implementación, pruebas y mantenimiento de un cluster: conocimientos básicos de hardware de computadoras, sistemas operativos y redes de computadoras. Por otra parte es necesario tener la habilidad para recabar información sobre tópicos especializados como componentes, controladores y librerías de interconexión de alta velocidad, reintroducirse en lenguajes de programación como C++ y librerías como MPI para empezar a escribir aplicaciones paralelas.

Debido a la gran aceptación en el mundo académico y científico, las características y funcionalidades más prominentes de los sistemas operativos, Linux es el sistema operativo más recomendado para disponer de un sistema cluster.

La instalación, configuración y administración de los recursos de un cluster podría tomar un tiempo considerable, como alternativa para reducir sustancialmente el tiempo requerido en la implementación de un cluster aparecen los toolkits, que son paquetes de software que automatizan el proceso de instalación, de configuración y de administración de un cluster completo. Estos toolkits, para instalación automática de clusters, pueden incluir una distribución de Linux; mientras que otros se instalan sobre una instalación existente de Linux. Sin embargo, incluso si primero se debe instalar Linux, los toolkits realizan la configuración e instalación de los paquetes requeridos por el cluster de forma automática. Del conjunto de toolkits existentes se pueden mencionar a NPACI Rocks [9] y a OSCAR [10].

III. DESARROLLO DE APLICACIONES PARALELAS

El paso de mensajes es el método más utilizado para programación de sistemas de memoria compartida distribuida. Este método puede ser sincrónico o asincrónico (llamadas bloqueantes o no bloqueantes). Es posible tener almacenamiento (copia del mensaje en un búfer) y hacer que se realicen otras tareas mientras se transmite un mensaje.

Para escribir programas paralelos, se puede utilizar compiladores basados en OpenMP [6], o librerías de paso de mensajes como PVM y MPI.

A. *Message Passing Interface*

En el año 1994 MPI emerge como un estándar de facto de computación paralela de memoria distribuida usando paso de mensajes. Esta interfaz intenta establecer un estándar práctico, eficiente, portátil y flexible para el paso de mensajes.

MPI no es un lenguaje de programación, es un conjunto de funciones y macros que conforman una librería estándar de C y C++, y subrutinas en Fortran. MPI ofrece un API, junto con especificaciones de sintaxis y semántica que explican como sus funcionalidades deben añadirse en cada implementación que se realice (tal como almacenamiento de mensajes o requerimientos para entrega de mensajes). Se han desarrrollado varias implementaciones de MPI basadas en la publicación del estándar [11], muchas de ellas son de libre distribución y algunas tienen limitaciones de portabilidad de código.

MPI incluye operaciones punto a punto, mediante llamadas bloqueantes (MPI_Send, MPI_Recv) y llamadas no bloqueantes (MPI_Isend y MPI_Irecv), y operaciones colectivas mediante llamadas de difusión (*broadcast*), recolección (*gather*), distribución (*scatter*) y reducción, otras llamadas colectivas que permiten obtener sincronización en una barrera (MPI_Barrier), y todas ellas destinadas a un grupo específico de procesos. MPI realiza la conversión de datos heterogéneos como parte transparente de sus servicios, por medio de la definición de tipos de datos específicos para todas las operaciones de comunicación; además, se pueden tener tipos de datos definidos por el usuario o primitivos.

IV. IMPLEMENTACIÓN DEL CLUSTER Y DESARROLLO DE UNA APLICACIÓN

A. *Implementación del Cluster Prototipo: aLeXAnDrE*

El presente artículo describe la implementación de dos clusters: uno basado en OSCAR y otro basado en NPACI Rocks. Ambos *clusters* están conformados por un nodo maestro y dos nodos esclavos, además ambos se derivan del proyecto Beowulf.

Se utilizó uno de los discos duros de cada computador personal para instalar el cluster basado en NPACI Rocks, y el otro disco duro para instalar el otro cluster basado en OSCAR.

El software utilizado se distribuye de forma libre. Se exploró el uso de varias distribuciones de Linux, entre ellas están: Red Hat 9.0, Mandrake 10.0, Fedora Core 3.0, y CentOS 4. Finalmente se instaló el primer cluster haciendo uso de Fedora Core 3.0, utilizando la herramienta de instalación automática de clusters OSCAR. En el segundo cluster se instaló CentOS 4.0 incluido en la herramienta de instalación automática de clusters NPACI Rocks.

Se realizaron pruebas sobre ambos sistemas. Las pruebas sobre el primer sistema OSCAR se basaron en las herramientas de pruebas que conforman el paquete de instalación. Se probaron los servicios Maui [12], NFS, PBS y la funcionalidad de las implementaciones de MPI que se proveen con el paquete: LAM/MPI y MPICH [13].

Sobre el sistema que tiene instalado Rocks se ejecutó el benchmark de Alto Rendimiento Linpack (HPL [14] – High Performance Linpack) que se incluye con su distribución. HPL es un software que permite resolver sistemas lineales aleatorios en aritmética de doble precisión. La ejecución de

HPL se realizó sobre cuatro procesadores (dos procesadores que utilizan tecnología HyperThreading). De esta prueba se obtuvo un rendimiento aproximado de 3.17 GFLOPS. Los resultados de las pruebas de benchmarking son utilizados para medir el rendimiento de supercomputadoras y generar la lista del Top500 [15] de supercomputadoras a nivel mundial.

TABLA I
COMPONENTES DE HARDWARE DE LOS NODOS DE ALEXANDRE

Componente	Descripción
Placa madre	Intel D865PERL
Procesador	Pentium IV de 3.0 GHz con tecnología HyperThreading
Memoria	512 MB DDR
Tarjeta de red	CNET 10/100/1000 Mbps ProG-2000S
Tarjeta de video	ATI Radeon 9250
Disco duro	2 discos SATA de 120 GB

B. Desarrollo de la Aplicación para Resolución de Cadenas de Markov

La aplicación se desarrolló utilizando MPI y permite resolver Cadenas de Markov. Las Cadenas de Markov ([16], [17]) pueden hacer uso de matrices de grandes dimensiones, por lo que su solución y las operaciones asociadas pueden requerir una gran capacidad de procesamiento computacional. Las operaciones más relevantes utilizadas en la solución de Cadenas de Markov son:

- Potencia de matrices.
- Resolución de sistemas de ecuaciones lineales.

Estas operaciones involucran un análisis previo para poder escribir un algoritmo que se adecue a las necesidades de compúto paralelo.

1) Operaciones con matrices y sistemas de ecuaciones lineales

En la fase de paralelización, cada operación deberá hacer que el proceso maestro calcule el número de filas que cada proceso esclavo va a utilizar:

$$Porción\ a\ procesar = Filas\ de\ matriz\ A / Número\ de\ procesos \quad (1)$$

a) Potencia de matrices.

El producto de una matriz A de l filas y m columnas, por una matriz B de m filas y n columnas es una matriz C, cuyos elementos están definidos por (2).

$$c_{i,j} = \sum_{k=0}^{m-1} a_{i,k} \cdot b_{k,j} \quad (2)$$

Mediante operaciones sucesivas basadas en las filas de una matriz se obtiene un algoritmo simple que permite efectuar la multiplicación de matrices. Este algoritmo se basa en un lazo, durante cada iteración del lazo basada en un índice i, se lee la fila i de la matriz A y los elementos de la columna j de la matriz B, y se escriben los datos en la fila i de la matriz C.

A partir de la multiplicación de matrices, se define la potencia n-ésima de una matriz A como:

$$C = A^n \quad (3)$$

$$C = (A \cdot A) \cdot A^{n-2} \quad (4)$$

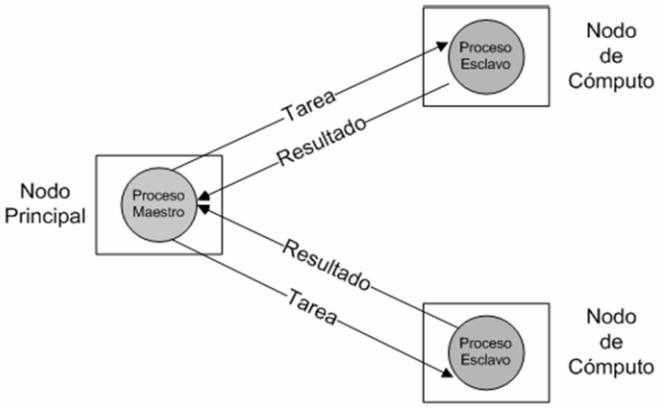

Fig. 1. En términos generales, la aplicación se diseñó en base al modelo Maestro–Esclavo. El proceso Maestro, el cual se ejecuta en el nodo principal, coordina las tareas realizadas por un grupo de procesos Esclavos, los cuales se encuentran en los nodos de cómputo.

En la operación potencia, cada elemento de la matriz C es una función de los elementos de la matriz A. Debido a que los valores de A no serán modificados en el algoritmo, se puede calcular de manera simultánea cada elemento de C. Para que la operación se realice de forma paralela el proceso maestro crea una copia de la matriz A denominada B.

El proceso maestro se encarga de envíar las filas de la matriz A correspondientes y toda la matriz B a cada uno de los procesos. Cada proceso es responsable de realizar la potencia de la porción asignada de filas de la matriz A usando la matriz B como la base de la operación potencia, generando los elementos de la matriz C correspondientes.

Después de realizar el procesamiento necesario para obtener la porción de la matriz A^n, los procesos envían esta porción de regreso al proceso maestro.

b) Resolución de sistemas de ecuaciones lineales.

Un conjunto de m ecuaciones con m incógnitas, puede representarse como:

$$A \cdot x = y \quad (5)$$

A se denomina matriz de conexión y contiene los coeficientes del sistema de ecuaciones, *x* se denomina vector incógnita y está conformado por las incógnitas de dicho sistema y *y* es el vector conocido, conformado por los términos independientes del sistema de ecuaciones.

El método de Gauss-Jordan consiste en la eliminación sucesiva de incógnitas. Se obtiene una matriz identidad mediante la sustitución de los elementos bajo y sobre la diagonal principal de la matriz de conexión por valores iguales a cero. Se logra mediante operaciones de suma y resta entre los elementos de las filas de la matriz y la utilización de una fila como pívot para realizar las operaciones correspondientes. Al obtener la matriz identidad, *y* (vector conocido) contendrá la solución del sistema de ecuaciones.

El procedimiento descrito anteriormente funciona de forma serial; para paralelizar el procedimiento, primero el maestro, envía la porción de filas de la matriz A y la porción de filas de la matriz *y* correspondientes a cada uno de los procesos esclavo. Cada proceso es responsable de normalizar la fila pívot que le corresponda y de reducir los elementos de las

otras filas; para esto, de forma ordenada, cada proceso envía la fila pívot a los otros procesos, es responsable de normalizarla y de hacer ceros los elementos de las otras filas que le fueron asignadas; los otros procesos realizarán las operaciones necesarias para que en base al pívot recibido, se encarguen de reducir sus filas. De esta forma, todos los procesos colaboran en la reducción de los elementos de la matriz A y obtienen los valores de la porción de la matriz *y* que les fue asignada. Una vez reducida la matriz A, cada proceso envía sus resultados (porción de la matriz *y*), al proceso maestro para que ensamble la matriz *y*.

Como se muestra en la Fig. 2, primero, el proceso P0, usa la fila 1 como pívot, la envía a los otros procesos, la normaliza y reduce la fila 2 en base al pívot. Los procesos P1 y P2 obtienen el pívot y reducen sus filas en base a este pívot. El proceso P0, procede a usar la fila 2 como pívot, la envía a los otros procesos, la normaliza y reduce la fila 1 en base al pívot; los otros procesos obtienen el pívot y reducen sus filas en base al pívot. Luego, el proceso P1 usa la fila 3 como pívot, la envía a los otros procesos, la normaliza y reduce la fila 4 en base a su pívot; los otros procesos obtienen el pívot y proceden a reducir sus filas. Y se continúa hasta que todos los procesos han reducido sus filas dando como resultado una matriz *y* que contiene los resultados parciales de la solución del sistema de ecuaciones.

2) Funcionalidad y estructura de la aplicación

La aplicación desarrollada permite obtener la distribución al paso *n* y la distribución de régimen de Cadenas de Markov de Tiempo Discreto (CMTD), o la distribución al tiempo *t* y la distribución de régimen de Cadenas de Markov de Tiempo Continuo (CMTC).

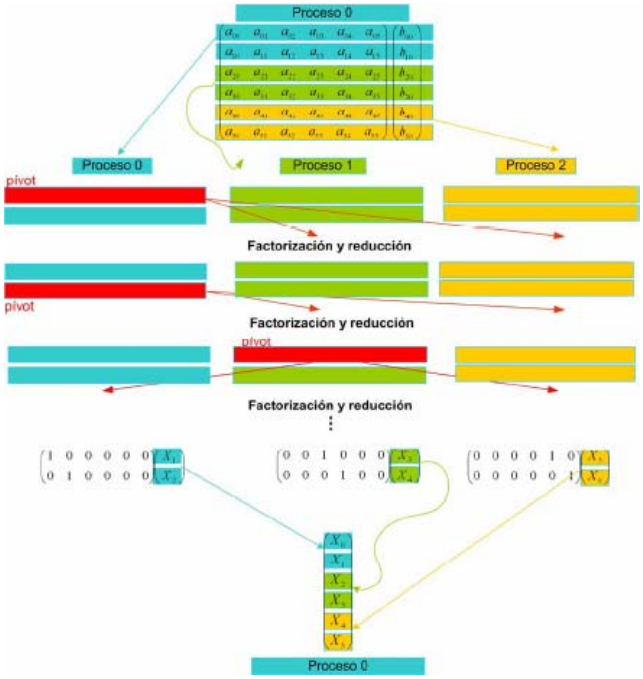

Fig. 2. Diagrama del proceso que se realiza para paralelizar la solución si se tiene una matriz A de 6 filas y 6 columnas y una matriz *y* de 6 filas y 1 columna, y existen 3 procesos, denominados P0, P1, P2; cada proceso será responsable de dar tratamiento a 2 filas de cada matriz. El proceso P0 será responsable de trabajar con la fila 1 y 2 de las matrices A y *y*; el proceso P1 será responsable de trabajar con la fila 3 y 4 de las matrices A y *y*; y el proceso P2 será responsable de trabajar con la fila 5 y 6 de las matrices A y *y*.

Utilizando C++ se diseñaron cuatro clases. La primera llamada CMatrix que define todas las operaciones secuenciales entre matrices. La segunda clase CMatrix_Handler que realiza las operaciones matriciales de forma paralela, y manipula las comunicaciones mediante el paso de mensajes (MPI). Las clases CMarkovTD y CMarkovTC permiten resolver las Cadenas de Markov de Tiempo Discreto y de Tiempo Continuo, respectivamente.

En síntesis, la aplicación realiza las siguientes funciones:

- El proceso maestro lee los datos de configuración y de la matriz P (CMTD) o Q(t) (CMTC).
- El proceso maestro inicializa la librería de MPI.
- El proceso maestro envía el paso al que se va a evaluar la CMTD o el tiempo al que se va a evaluar la CMTC, y la dimensión de la matriz P o Q(t) a los otros procesos.
- Se obtiene la distribución al paso n o al tiempo t. Para lo cual el proceso maestro calcula la porción de trabajo que le corresponde realizar a cada proceso y envía la porción correspondiente de la matriz (P o Q(t)) para que los otros procesos ayuden en las operaciones requeridas. Luego de realizar las operaciones, los procesos envían sus resultados al proceso maestro para que los presente en consola y los almacene en un archivo.
- Se obtiene la distribución de régimen. Para lo cual el proceso maestro forma la matriz de conexión y el vector conocido, envía las porciones correspondientes a cada proceso para su resolución y luego recupera los resultados para presentarlos en consola y almacenarlos en un archivo.
- Cada proceso libera la memoria utilizada y el proceso maestro se encarga además de liberar las librerías de MPI.

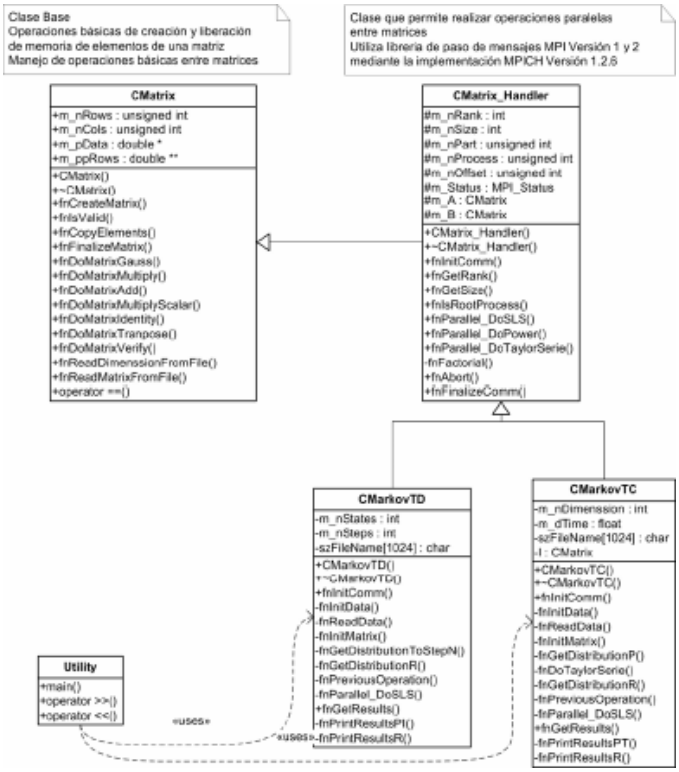

Fig. 3. Diagrama UML de clases y la interrelación entre las clases diseñadas y sus principales funciones de la aplicación.

Finalmente, se diseñó una aplicación con GUI, presentada en la Fig. 4, que permite ejecutar la aplicación desarrollada. Se utilizó Qt Designer [18], una herramienta que permite diseñar e implementar interfaces con un conjunto de herramientas multiplataforma. Qt es una librería de clases de C++ que provee portabilidad de código, se ejecuta sobre varios sistemas operativos como Windows, Mac OS X, Linux y Unix.

A la aplicación con GUI se le denominó SolVer, ésta permite recuperar los datos desde un archivo de texto, el cual contiene los elementos de la matriz P o la matriz Q(t); además, almacena el número de pasos o el tiempo en el que se evalúa la Cadena de Markov. Permite ejecutar la aplicación paralela con la cantidad de procesos especificada mediante el control deslizador (slider), obtener el resultado del cómputo paralelo y presentarlo en pantalla.

Fig. 4. SoLVeR GUI para ejecutar la aplicación paralela.

3) Análisis de resultados

La aplicación fue ejecutada utilizando un solo computador (1 procesador), y sobre el cluster usando 2 computadoras (2 procesadores) y 3 computadoras (3 procesadores).

Para probar la funcionalidad de la aplicación, se realizaron pruebas con matrices de orden 3, 30, 60, 90, 120, 150, 210, 270, 330, 390, 450, 510 y 600. Se realizaron pruebas resolviendo Cadenas de Markov a Tiempo Discreto y a Tiempo Continuo.

Para evaluar las Cadenas de Markov a Tiempo Discreto y a Tiempo Continuo, se obtuvo la distribución en el paso 25 y la distribución en el tiempo 0,25, respectivamente, y la distribución de régimen para ambas. Se registró el tiempo de ejecución de la aplicación, el tiempo que tardó en obtener las distribuciones en el paso 25 y en el tiempo 0,25, y el tiempo que tomó en calcular la distribución de régimen en ambos casos.

En la Fig. 5 y Fig. 6 se puede observar que en promedio, usando 2 procesadores, el tiempo que toma obtener la distribución en el paso 25 de Cadenas de Markov a Tiempo Discreto y en el tiempo 0,25 de Cadenas de Markov a Tiempo Continuo, es un valor cercano a la mitad del tiempo que toma el hacerlo con un solo procesador; y usando 3 procesadores, el tiempo que toma se reduce a un valor cercano al tercio del valor que toma sobre uno sólo.

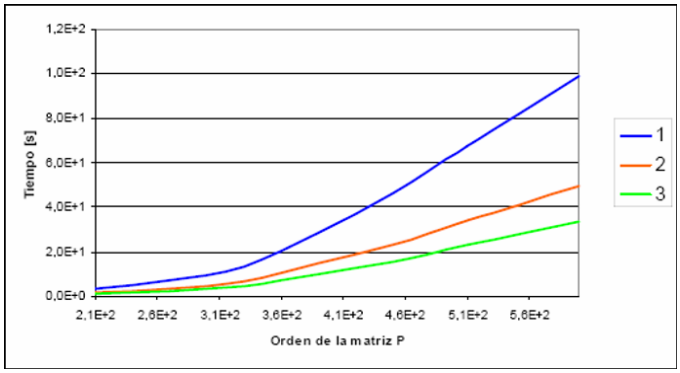

Fig. 5. Comparación de los resultados obtenidos al calcular la distribución en el paso 25 con 1, 2 y 3 procesadores.

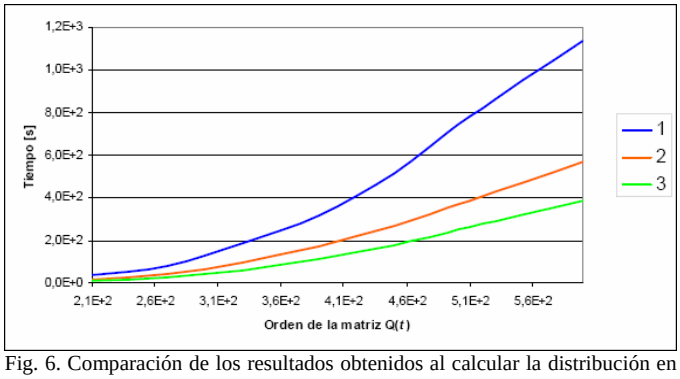

Fig. 6. Comparación de los resultados obtenidos al calcular la distribución en el tiempo 0.25 con 1, 2 y 3 procesadores.

De la Fig. 7 y Fig. 8 se puede concluir que en el caso de la distribución de régimen, tanto en CMTD como en CMTC, se tienen valores que no muestran una mejora al compararlos con respecto a los valores obtenidos en un procesador, para matrices de tamaño pequeño (de orden 3 a 150), debido a que la cantidad de operaciones es baja comparada con el tiempo requerido para el envío de datos entre procesos. A pesar de ello se puede apreciar una mejora en las matrices de mayor tamaño (orden superior a los 150), usando 2 procesadores el tiempo que se tarda en obtener la solución del sistema de ecuaciones se reduce aproximadamente en un 50%, y utilizando 3 procesadores el tiempo se reduce en promedio en un 60%.

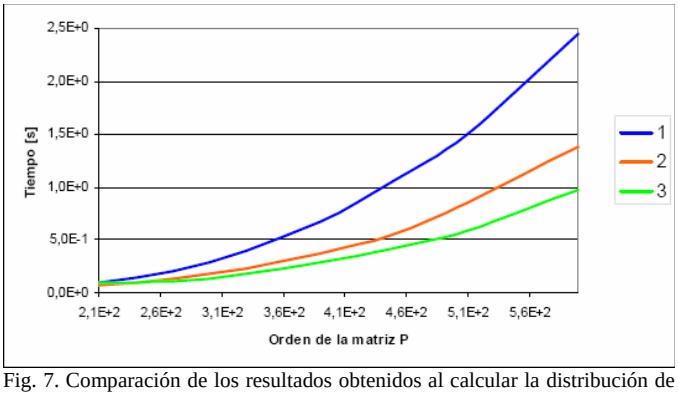

Fig. 7. Comparación de los resultados obtenidos al calcular la distribución de régimen con 1, 2 y 3 procesadores.

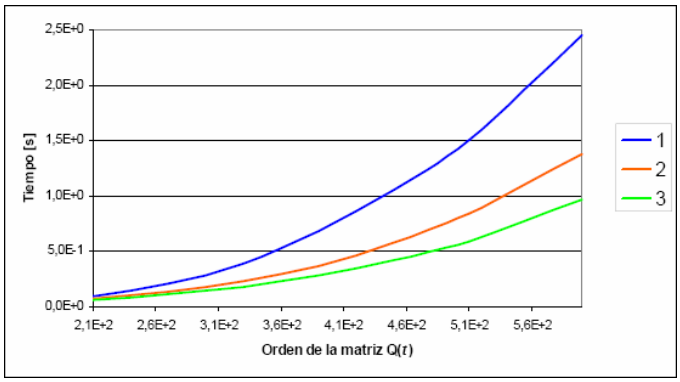

Fig. 8. Comparación del tiempo requerido con 1, 2 y 3 procesadores para obtener la distribución de régimen (CMTC).

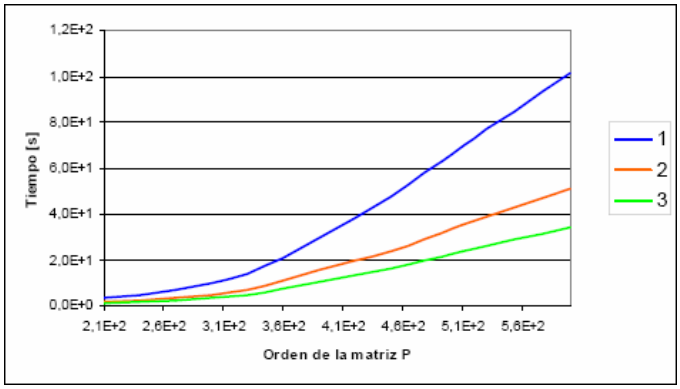

Fig. 9. Comparación de los tiempos de ejecución que tomó la resolución de la CMTD.

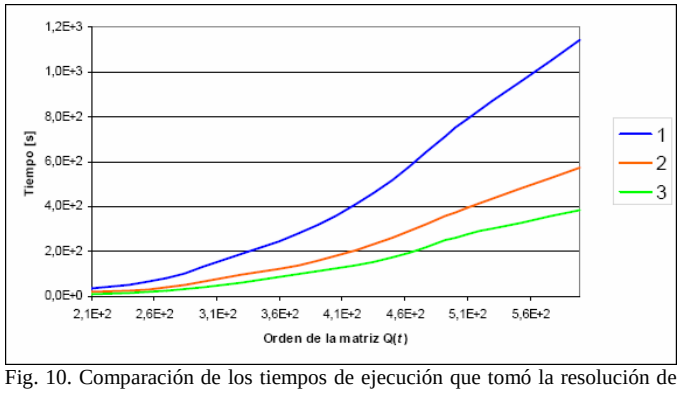

Fig. 10. Comparación de los tiempos de ejecución que tomó la resolución de la CMTC.

Finalmente, se puede mencionar que el tiempo de ejecución total se comporta de forma similar a la descrita para la distribución en el paso 25 o en el tiempo 0.25, como se puede obstar en la Fig. 9 y Fig. 10. En general, observando las figuras presentadas, se puede mencionar que el tiempo en leer la matriz P o Q(t), y almacenar los resultados, es decir las operaciones no paralelizadas, consumen poco tiempo comparado con las operaciones realizadas en paralelo.

4) Extensión de análisis de resultados

En MPI se puede evaluar el tiempo estimado mediante la utilización de MPI_Wtime. Esta función se deberá llamar cada vez que se requiere conocer el tiempo consumido en una cierta sección del programa, antes y después de la ejecución. De esta manera se pudieron obtener los tiempos estimados de ejecución de cada una de las operaciones concernientes a las Cadenas de Markov.

Por otra parte MPI provee un interfaz de evaluación del rendimiento de aplicaciones paralelas conocido como MPE [19] (Multi-Proccessing Enviroment). Al enlazar el código fuente con la librería de MPE se obtiene información del tiempo consumido por el programa en cada llamada que se realiza a MPI. MPE genera automáticamente archivos de registro (log) para luego ser interpretados mediante la utilización de herramientas de visualización como: Upshot, Nupshot, Jumpshot-2, 3, y 4 [20].

Estas herramientas despliegan información de la ejecución de aplicación mediante la utilización de diagramas de Gantt. En el diagrama de Gantt se representa a cada proceso mediante una línea de tiempo, y cada línea de tiempo muestra el estado de dicho proceso.

En el sistema aLeXAnDrE se utilizó MPE para visualizar los archivos generados al ejecutar la aplicación de resolución de Cadenas de Markov sobre Rocks y OSCAR.

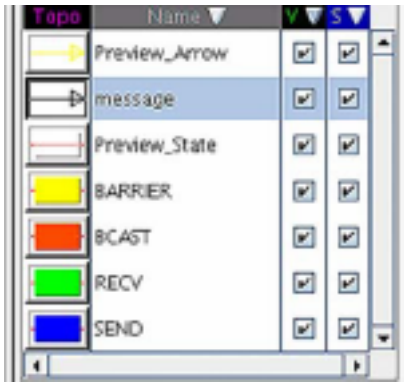

Fig. 11. Leyenda que indica el tipo de llamada a MPI la cual se identifica por un color específico. En las siguientes figuras se utiliza la simbología para message, BARRIER, BCAST, RECV y SEND; y no se utilizan Preview_Arrow y Preview_State. BARRIER muestra el tiempo que toma una llamada a MPI_Barrier, BCAST muestra el tiempo que toma una llamada a MPI_Bcast, RECV presenta el tiempo que tarda realizar una llamada a MPI_Recv y SEND indica el tiempo de la llamada MPI_Send.

Como se puede apreciar en la Fig. 12 y Fig. 13, se dispone de tres procesos: un maestro y dos procesos esclavos. En la primera fase los procesos 1 y 2 consumen una gran cantidad de tiempo en la primera llamada a MPI_Bcast, debido a que deben esperar a que el proceso maestro realice operaciones locales y el envio de parámetros para evaluar las Cadenas de Markov. Luego el proceso maestro realiza una llamada MPI_Send para enviar la porción de datos correspondiente a cada proceso, y los otros dos procesos realizan la llamada a MPI_Recv para recibir los datos. Se transmiten todos los elementos de la matriz mediante MPI_Bcast para obtener la distribución al paso. El proceso maestro realiza una llamada a MPI_Recv para recibir los datos de resultados desde los otros procesos. Finalmente se realiza una llamada a MPI_Barrier.

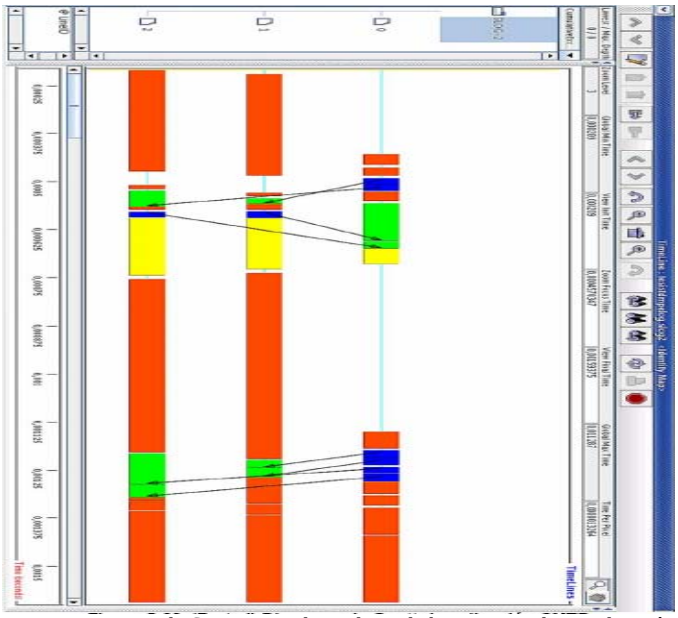

Fig. 12. Diagrama de Gantt generado mediante la herramienta de visualización Jumpshot-4 que incluye Rocks. En esta figura se pueden ver las diferentes llamadas a MPI que realiza la aplicación para resolver la CMTD especificadas por la leyenda de la Fig. 11.

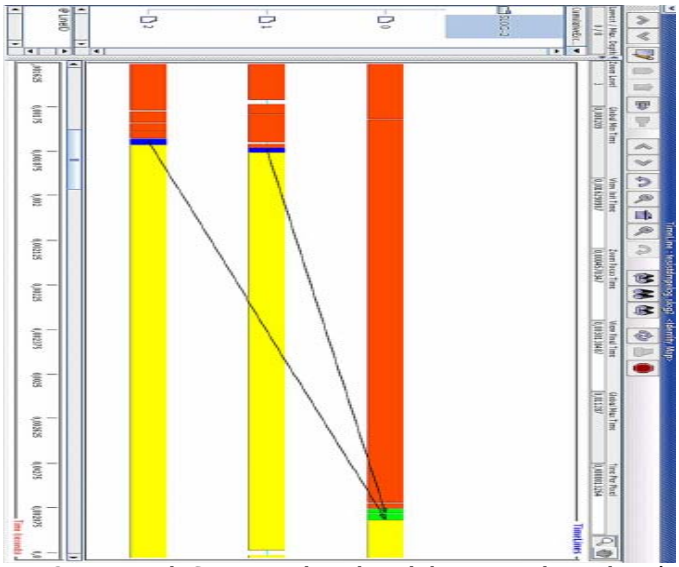

Fig. 13. Diagrama de Gantt generado mediante la herramienta de visualización Jumpshot-4 que incluye Rocks. Se pueden ver las diferentes llamadas a MPI que realiza la aplicación para resolver la CMTD especificadas por la leyenda de la Fig. 11.

Los procesos 1 y 2 consumen más tiempo en la llamada a MPI_Barrier debido a que el proceso maestro debe recibir los resultados antes de realizar esta llamada. Una vez que todos los procesos alcanzan la barrera de sincronización, el proceso maestro realiza operaciones locales (escritura de resultados), y los otros procesos esperan por la finalización de MPI_Bcast. Luego el proceso maestro envía las partes correspondientes de las matrices que permiten resolver el sistema de ecuaciones lineales, primero al proceso 1 y luego al proceso 2. Se realizan una serie de llamadas a MPI_Bcast para enviar el pívot y realizar las operaciones correspondientes. El proceso maestro realiza la llamada a MPI_Recv para recibir los resultados emitidos por los procesos 1 y 2, los cuales realizan una llamada a MPI_Send. El proceso maestro recibe primero los resultados del proceso 1 y luego de finalizada esta operación recibe los resultados del proceso 2. El proceso maestro, y los esclavos 1 y 2 realizan una llamada a MPI_Barrier.

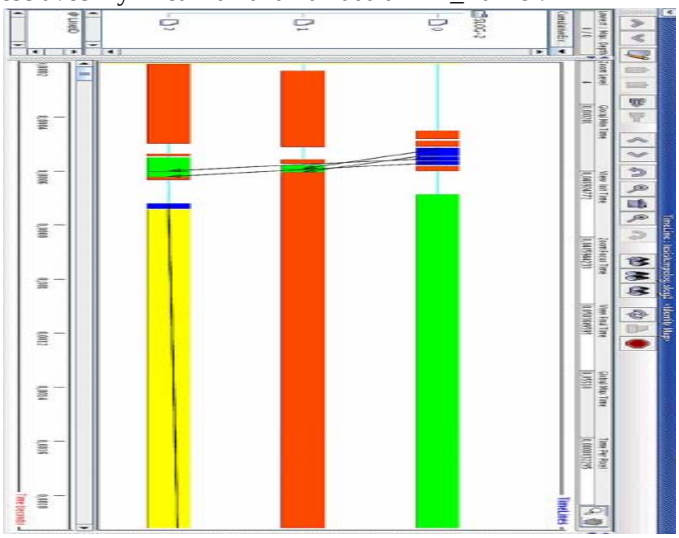

Fig. 14. En la figura se pueden ver las diferentes llamadas a MPI que realiza la aplicación para resolver la CMTC especificadas por la leyenda de la Fig. 11.

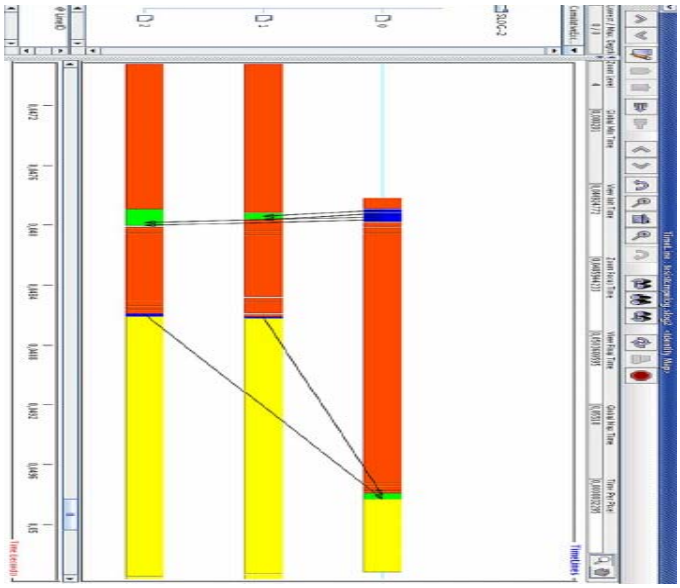

Fig. 15. En la figurase pueden ver las diferentes llamadas a MPI que realiza la aplicación para resolver la CMTC especificadas por la leyenda de la Fig. 11.

El procedimiento es similar en la obtención de la solución de Cadenas de Markov CMTC, como se muestra en la Fig. 14 y Fig. 15, donde se pueden apreciar llamadas a MPI_Bcast, MPI_Send, MPI_Recv y MPI_Barrier y lo descrito en los párrafos anteriores.

De estos gráficos se puede concluir, que el tiempo que toma una llamada de MPI incluye los tiempos de otras operaciones que se realizan con el mensaje: encripción, transmisión y desencripción, y cuya duración es de un valor relativamente pequeño; los procesos 1 y 2 consumen una gran cantidad de tiempo en algunas de las llamadas a MPI_Bcast, debido a que deben esperar a que el proceso maestro realice operaciones seriales (locales), como lectura de la matriz P o Q(*t*) desde un archivo de texto, impresión en consola y almacenamiento de resultados en archivos.

V. CONCLUSIÓN

Los clusters se presentan como una solución económica viable para obtener una gran capacidad computacional. En junio de 2005, el 60,8% de los 500 computadoras más veloces del mundo eran clusters. En nuestro país existe una gran cantidad de equipos que se dan de baja o se dejan de usar por que se consideran obsoletos; sin embargo, podrían usarse para construir un cluster que presente características de procesamiento realmente buenas, y el costo involucrado en la construcción sería relativamente bajo.

La aplicación desarrollada fue concebida con la intención de demostrar las bondades del cluster. Utilizando el lenguaje C++ y la librería de paso de mensajes MPI, esta aplicación permite resolver Cadenas de Markov que involucran resolución de sistemas de ecuaciones, y operaciones entre matrices (multiplicación, potencia), haciendo uso de la capacidad computacional que el cluster ofrece.

El presente artículo presenta la experiencia obtenida en el desarrollo de un proyecto sobre clusters que espera ser la base para futuros trabajos que traten de buscar soluciones más avanzadas para producir clusters de mejores características, haciendo uso de mejores herramientas.

REFERENCIAS

[1] http://www.answers.com/topic/massively-parallel
[2] Buyya, R. "High Performance Cluster Computing: Architectures and Systems", vol. 1, Upper Saddle River, Ed. New Jersey: Prentice Hall, 1999, pp. 15-64.
[3] Culler, D. y Singh J., "Parallel Computer Architectures: A hardware/Software Approach", Paint Street, Ed. San Francisco: Morgan Kaufmann Publishers, Inc, 1999.
[4] Buyya, R. "High Performance Cluster Computing: Programing and Applications", vol. 2, Upper Saddle River, Ed. New Jersey: Prentice Hall, 1999, pp. 15-64.
[5] Pacheco P, "Parallel Programming with MPI", Paint Street, Ed. San Francisco: Morgan Kaufmann Publishers, Inc, 1997.
[6] Quinn M, "Parallel Programming in C with MPI and OpenMP", Ed. New York: McGraw-Hill, 2003.
[7] Lucke R, "Building Clustered Linux Systems", Upper Saddle River, Ed. New Jersey: Prentice Hall, 2005.
[8] Pfister G., "In Search of Clusters: The Coming Battle in Lowly Parallel Computing", Upper Saddle River, Ed. New Jersey: Prentice Hall, 1995.
[9] http://www.rocksclusters.org
[10] http://oscar.openclustergroup.org/
[11] http://www-unix.mcs.anl.gov/mpi/implementations.html
[12] http://clusterresources.com/products/maui
[13] http://www-unix.mcs.anl.gov/mpi/mpich
[14] http://www.netlib.org/benchmark/hpl/tuning.html
[15] http://www.top500.org
[16] Norris, J. Markov Chains. Universidad de Cambridge, Inglaterra, 1998.
[17] http://www.cms.wisc.edu/~cvg/course/491/modules/Markov/Markov/node2.html
[18] Blanchette, J. y Summerfield, M.,: "C++ GUI Programming with Qt 3". Ed. Estados Unidos: Pearson Education, Inc., Primera Edición, 2004.
[19] (Manual de usuario y fuente en línea) Chan A. y Gropp W. (2005, Mayo, 26), "User's Guide for MPE: Extensions for MPI" [Documento Adobe Acrobat]. Laboratorio Nacional Argonne, Estados Unidos. Disponible http://www-unix.mcs.anl.gov/perfvis/software
[20] (Manual de usuario y fuente en línea) Chan A. y Gropp W. (2005, Mayo, 26), "User's Guide for Jumpshot-4" [Documento Adobe Acrobat]. Laboratorio Nacional Argonne, Estados Unidos. Disponible http://www-unix.mcs.anl.gov/perfvis/software/viewers/jumpshot-4/usersguide.html

Raúl D. Mejía

Ingeniero en Electrónica y Redes de Información, Escuela Politécnica Nacional (EPN) en Quito-Ecuador en 2005. Certificación Linux ACE Advance Career de IBM, Universidad Técnica Particular de Loja en Quito-Ecuador en 2005. Actualmente es instructor de la Academia Linux ACE de la Universidad Técnica Particular de Loja de la sede Quito.

Diego A. Fernández

Ingeniero en Electrónica y Redes de Información, Escuela Politécnica Nacional (EPN) en Quito-Ecuador en 2005. Certificación Linux ACE Advance Career de IBM, Universidad Técnica Particular de Loja en Quito-Ecuador en 2005. Actualmente es instructor de la Academia Linux ACE de la Universidad Técnica Particular de Loja de la sede Quito.

Iván M. Bernal

Ingeniero en Electrónica y Telecomunicaciones, Escuela Politécnica Nacional (EPN), en Quito-Ecuador en 1992. Obtuvo los títulos de M.Sc. (1997) y Ph.D. (2002) en Computer Engineering en Syracuse University, NY, USA. Actualmente es docente de la EPN, en el Departamento de Electrónica, Telecomunicaciones y Redes de Información.