Proposición de un método para balanceo de carga en un cluster heterogéneo simulado en NS2

Proposed method for load balancing in heterogeneous cluster simulated in NS2

John William Branch Bedoya[1], Ph.D., Félix Francisco Ramos Corchado[2], Ph.D., Andrea Mesa Múnera[1], MSc. (c) y Raúl Esteban Jiménez Mejía[1].

1. Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV)– Unidad Guadalajara, México

Universidad Nacional de Colombia Sede Medellín

framos@gdl.cinvestav.mx, jwbranch@unalmed.edu.co, amesamu@unal.edu.co, rejimene@unalmed.edu.co

Recibido para revisión 10 de enero de 2009, aceptado 21 de mayo de 2009, versión final 28 de mayo de 2009

Resumen-Este trabajo tiene como objetivo principal analizar la importancia que tiene en la actualidad el procesamiento paralelo y distribuido cuando nos enfrentamos a problemas potenciales en ingeniería. Se propone la implementación de un algoritmo que involucre un nuevo método para balancear la carga de trabajo en un cluster heterogéneo teniendo en cuenta la aptitud que tiene cada uno de los nodos para ejecutar ciertas tareas, y así, poder realizar la asignación de la carga de acuerdo a la capacidad de cada nodo minimizando el tiempo de ejecución total.

Palabras Clave-Hardware, Arquitecturas Paralelas y Sistemas Operativos, Sistemas Distribuidos, Redes y Teleinformática, Cluster, Balanceo de Carga, Procesamiento Paralelo y Computación Distribuida.

Abstract-The purpose of this work is to analyze the importance of the current parallel and distributed processing when faced with potential problems in engineering. It proposes the implementation of an algorithm that involves a new method to balance the workload in a heterogeneous cluster taking into account the capacity of each node to perform certain tasks, so that make the allocation of the load according to the capacity of each node by minimizing the total execution time.

Keywords- Hardware, Parallel Architectures and Operating Systems, Distributed Systems, Networks and Teleinformation, Cluster, Load Balancing, Parallel Processing and Distributed Computing.

I. INTRODUCCIÓN

Con el pasar de los años el hombre se ha topado con diversos problemas que no han sido fáciles de resolver con máquinas simples, por lo que él mismo se ha visto en la necesidad de frecuentar herramientas para facilitar su solución.

Para disipar estos problemas nacieron los llamados supercomputadores, los cuales cuentan con arreglos de microprocesadores que trabajan en sincronía empleando procesamiento paralelo. A los supercomputadores se les conoce hoy en día como cluster de computadores, concepto no muy difundido hasta hace poco tiempo, pero que en la actualidad es de gran ayuda en la computación paralela y distribuida logrando ayudar al hombre con soluciones más óptimas frente a problemas robustos.

Un cluster se puede definir como un grupo de múltiples computadores unidos por una red de alta velocidad, de tal forma que el conjunto puede ser visto como una única máquina, pero que por su poder de cómputo resuelve problemas que un solo equipo de escritorio no podría hacer.

Los cluster deben contener diversas características que hacen que sean viables en el manejo de grandes cantidades de tareas. Tienen como finalidad agrupar el poder de cómputo de los nodos implicados para proporcionar una mayor escalabilidad, disponibilidad y fiabilidad [1] y [2]. La escalabilidad es la capacidad de un equipo para hacer frente a volúmenes de trabajo cada vez mayores sin, por ello, dejar de prestar un nivel de rendimiento aceptable.

La disponibilidad y la fiabilidad, se encuentran bastante relacionados, aunque difieren ligeramente en algunos aspectos. La disponibilidad es la calidad de estar presente y listo para su uso, mientras que la fiabilidad es la probabilidad de un correcto funcionamiento [1].

Actualmente existen algunas variables que afectan estas características, las cuales se encuentran directamente ligadas al rendimiento que un cluster puede presentar o a la forma en que trabaja éste (migración de procesos, latencia de red, balanceo de carga, gestión de memoria y monitorización); en este trabajo se

enfocará en el manejo del balanceo de carga, mediante el cual se pretende reducir la diferencia de carga entre pares de nodos, migrando procesos desde los nodos más cargados hacia los menos cargados.

De [3], el balanceo de carga trata de distribuir la carga de trabajo de acuerdo a la disponibilidad de procesamiento y a los recursos de cada equipo en un sistema computacional. Esta distribución pretende maximizar la utilización de los recursos, posibilitando el mejor desempeño del sistema.

En la actualidad el problema del balanceo de carga ha sido bastante trabajado, debido al hecho de que las diferentes comunidades requieren disminuir al máximo el tiempo de ejecución de las aplicaciones ejecutadas. Sin embargo, su solución no es siempre trivial. Este es un problema bastante complejo, debido a la dificultad en ocasiones de lograr que las propuestas en las distribuciones de carga de trabajo sean fácilmente escalables, o que se pueda trabajar con equipos que no sean necesariamente similares; en general se recurre a trabajar con cluster homogéneos (maquinas con software ó hardware semejantes) y no en heterogéneos (maquinas con software ó hardware diferentes).

El objetivo general de este trabajo es entonces, proponer un método para efectuar el manejo del balanceo de carga en sistemas de cómputo distribuido de alto desempeño, empleando técnicas propias de la computación paralela por medio del uso de un cluster heterogéneo.

En el marco de esta investigación se tiene como prioridad plantear este método siendo éste aporte valioso para la comunidad científica local.

Aunque la parte de implementación es importante para la investigación ésta sólo alcanza un papel secundario para mostrar únicamente los resultados encontrados durante el proceso. La validación fue efectuada en un simulador de tiempo discreto llamado Network Simulator-2 (NS-2), en donde fue implementado el algoritmo de balanceo. Los nodos utilizados que conformaran el cluster son equipos que contaran únicamente con hardware diferente.

El NS-2 se apoya en dos lenguajes de programación para su correcto funcionamiento: OTcl y C++. Como resultado de la simulación se obtienen gran cantidad de datos matemáticos para los estudios posteriores y trazas específicas que son visualizadas en la herramienta NAM del ns.

Los resultados obtenidos muestran que el método propuesto alcanza todas las expectativas logrando mejorar los tiempos de ejecución en comparación con los métodos con los que fue comparado.

El presente artículo está organizado de la siguiente manera: en primer lugar se presentará una revisión de los conceptos y términos fundamentales de los cluster de computadores y algunos trabajos previos aplicativos. Luego se efectuará una contextualización y trabajos relacionados al estado del arte sobre el balanceo de carga. Posteriormente se dará a conocer el proceso utilizado para la implementación del algoritmo de balanceo de carga. Finalmente se presentará: el análisis de resultados, las conclusiones y trabajo futuro.

To insert images in Word, position the cursor at the insertion point and either use Insert | Picture | From File or copy the image to the Windows clipboard and then Edit | Paste Special | Picture (with "Float over text" unchecked).

IEEE will do the final formatting of your paper. If your paper is intended for a conference, please observe the conference page limits.

II. CLUSTER DE COMPUTADORES

A. Conceptos y Términos Fundamentales

La ejecución de tareas complejas siempre ha sido un problema en el mundo, ya que se debe contar con el equipo necesario para la elaboración y solución de estas.

En el pasado gran cantidad de aplicaciones no podían ser ejecutadas debido a la deficiencia de computo de las máquinas de esa época. Se hacia necesario el uso de grandes equipos que funcionaran sólo para tal fin, estos equipos eran denominados supercomputadores. Sin embargo este tipo de computadores no era accesible para cualquiera ya que su costo era bastante elevado.

Tras la necesidad de solucionar el problema de complejidad en las tareas y el elevado costo de los equipos que pudieran resolver dichas aplicaciones nacen los cluster.

El concepto de Cluster nació cuando los pioneros de la supercomputación intentaban difundir diferentes procesos entre varios computadores, para luego poder recoger los resultados que dichos procesos debían producir. Con un hardware más barato y fácil de conseguir se pudo perfilar que podrían conseguirse resultados muy parecidos a los obtenidos con aquellas máquinas mucho más costosas [4].

Según [5], el uso de clusters para desarrollar, depurar y ejecutar aplicaciones paralelas está ganando en popularidad, convirtiéndose en una gran alternativa al uso de arquitecturas más especializadas debido al gran precio de estas. Un factor importante que ha hecho que el uso de clusters sea práctico es la estandarización de numerosas herramientas y utilidades usadas en aplicaciones paralelas.

B. Clasificación de los Cluster

Existen principalmente tres tipos de cluster: cluster de alto rendimiento, cluster de alta disponibilidad y cluster de alta confiabilidad, los cuales fueron creados para desempeñar funciones especificas, de acuerdo a los requerimientos y preferencias de quien o quienes los estén implementando.

1) Alto rendimiento (HP, high performance): Según [6], los clusters de alto rendimiento han sido creados para compartir el recurso más valioso de un computador, el tiempo de proceso. Cualquier operación que necesite altos tiempos de CPU puede ser utilizada en un cluster de alto rendimiento, siempre que se encuentre un algoritmo que sea paralelizable.

2) Alta disponibilidad (HA, high availability): Según [6], los clusters de alta disponibilidad son bastante ortogonales en lo que se refieren a funcionalidad a un cluster de alto rendimiento. Los clusters de alta disponibilidad pretenden dar servicios 7/24 de cualquier tipo, son clusters donde la principal funcionalidad es estar controlando y actuando para que un servicio o varios se encuentren activos durante el máximo periodo de tiempo posible.

3) Alta confiabilidad (HR, high reliability): Según [6], estos clusters tratan de aportar la máxima confiabilidad en un entorno, en la cual se necesite saber que el sistema se va a comportar de una manera determinada. Puede tratarse por ejemplo de sistemas de respuesta a tiempo real.

Este tipo de clusters son los más difíciles de implementar. No se basan solamente en conceder servicios de alta disponibilidad, sino en ofrecer un entorno de sistema altamente confiable. Esto implica muchísima sobrecarga en el sistema.

C. Trabajos Previos Sobre Cluster

De [7], en 1994, se integró el primer cluster de computadores en el Centro de Vuelos Espaciales Goddard de la NASA, para resolver problemas computacionales que aparecen en las ciencias de la tierra y el espacio. Los pioneros de este proyecto fueron Thomas Sterling, Donald Becker y otros científicos de la NASA. El cluster de PCs desarrollado tuvo una eficiencia de 70 megaflops (millones de operaciones de punto flotante por segundo). Los investigadores de la NASA le dieron el nombre de Beowulf a este cluster.

En 1996, hubo también otros dos sucesores del proyecto Beowulf de la NASA. Uno de ellos es el proyecto Hyglac desarrollado por investigadores del Instituto Tecnológico de California (CalTech) y el Laboratorio de Propulsión Jet (JPL), y el otro, el proyecto Loki construido en el Laboratorio Nacional de Los Alamos, en Nuevo México. Cada cluster se integró con 16 microprocesadores Intel Pentium Pro y tuvieron un rendimiento sostenido de más de un gigaflop con un costo menor a $50,000 dólares.

Debido al elevado potencial de procesamiento paralelo y distribuido que los cluster ofrecen, muchos grupos de investigación han buscado soluciones que permitan el máximo aprovechamiento de estos [8]. Hoy en día, la existencia de superordenadores que trabajen en tiempo real, se ha convertido en una necesidad [9]. Lo que se busca realmente en la actualidad, es que los nuevos supercomputadores alcancen velocidades cada vez mayores con el fin de que se puedan solucionar fácilmente diversas aplicaciones del mundo real.

En 1999, Hoganson [10] trabaja en un modelo analítico para analizar el desempeño de clusters interconectados, el cual explora diferentes estrategias de localización de procesos en procesadores en un ambiente multitareas. Haciendo uso de heurísticas, el modelo presenta la mejor relación entre tamaño del cluster en aplicaciones paralelas dedicadas a cada segmento del cluster aumentando el paralelismo y reduciendo la sobrecarga de la comunicación. Tres años después, Nguyen y Peirre [11] también crearon un modelo analítico de una experimentación en un simulador de un sistema de archivos paralelos analizando básicamente el factor de la escalabilidad de los cluster. Como resultado, se mostró que el crecimiento de un cluster depende del grado de saturación de redes de comunicación. La saturación de redes de comunicación limita la escalabilidad del cluster.

Andresen y sus compañeros en el 2003 [12] presentaron un sistema de monitoreo de comunicación en cluster Beowulf. El sistema fue bautizado DISTOP y puede ser utilizado a nivel de proceso con bajo consumo de recursos.

En el 2005 Avresky y Natchev [13] trabajaron en un algoritmo de reconfiguración dinámica de redes para tolerar fallas con múltiples nodos y enlaces, especialmente en redes de alta velocidad con topología arbitraria. En el mismo año Haase y sus compañeros trabajaron en el SDVM un cluster de computadores heterogéneos, que permite operar en cualquier topología de red y con capacidad de expansión [14].

III. BALANCEO DE CARGA

A. Conceptos y Términos Fundamentales

De [15], el aspecto del balanceo de carga es bastante importante debido a que en muchas aplicaciones paralelas, como la búsqueda o la optimización, es extraordinariamente difícil predecir el tamaño de las tareas asignadas a cada procesador, de manera que se realice una división de las mismas para que todos mantengan la carga computacional uniforme. Cuando no es uniforme, es decir, hay desbalanceo en la carga, entonces algunos procesadores terminarán permaneciendo inactivos mientras otros todavía están calculando.

Frecuentemente todos los procesadores o algunos de ellos necesitan algún sincronismo durante la ejecución del programa, de forma que si no están todos en la misma situación en todo instante entonces algunos de ellos deberán esperar a que otros terminen.

El estudio del balanceo de carga es muy importante para poder distribuir de una forma equitativa la carga computacional entre todos los procesadores disponibles y con ello conseguir la máxima velocidad de ejecución.

Sin embargo, puede ocurrir que algunos procesadores finalicen sus tareas antes que el resto y queden libres debido a que el trabajo no se haya repartido de una forma equitativa o porque algunos procesadores sean más rápidos que otros, o por ambas situaciones. La situación ideal es que todos los procesadores trabajen de una forma continua sobre las tareas disponibles para conseguir el mínimo tiempo de ejecución.

B. Tipos de Balanceo de Carga

Según [15], existen dos formas de balanceo de carga: balanceo de carga estático y balanceo de carga dinámico. En el primer caso, la distribución de las tareas se realiza al comienzo de la computación, lo cual permite al maestro (nodo principal dentro del cluster) participar en la computación una vez que haya asignado una fracción del trabajo a cada esclavo (el resto de los nodos de un cluster. Los nodos esclavos obedecen órdenes del nodo maestro). La asignación de tareas se puede realizar de una sola vez o de manera cíclica. El segundo caso, balanceo de carga dinámico es muy útil cuando el número de tareas es mayor que el número de procesadores disponibles o cuando el número de tareas es desconocido al comienzo de la aplicación. Una importante característica del balanceo de carga dinámico es la capacidad que tiene la aplicación de adaptarse a los posibles cambios del sistema, no sólo a la carga de los procesadores sino también a posibles reconfiguraciones de los recursos del sistema. Debido a esta característica, un cluster puede responder bastante bien cuando se produce el fallo de algún procesador, lo cual simplifica la creación de aplicaciones tolerantes a fallos que sean capaces de sobrevivir cuando se pierde algún esclavo o incluso el maestro.

1) Balanceo de Carga Estático: De [15], el balanceo de carga también es llamado mapeado del problema o planificación del problema. Este tipo de balanceo de carga se trata antes de la ejecución de cualquier proceso.

El balanceo de carga estático tiene serios inconvenientes que lo sitúan en desventaja sobre el balanceo de carga dinámico. Entre ellos cabe destacar los siguientes:

- Es muy difícil estimar de forma precisa el tiempo de ejecución de todas las partes en las que se divide un programa sin ejecutarlas.
- Algunos sistemas pueden tener retardos en las comunicaciones que pueden variar bajo diferentes circunstancias, lo que dificulta incorporar la variable retardo de comunicación en el balance de carga estático.
- A veces los problemas necesitan un número indeterminado de pasos computacionales para alcanzar la solución.

2) Balanceo de Carga Dinámico: Según [15], este tipo de balanceo de carga se trata durante la ejecución de procesos.

Con el balanceo de carga dinámico todos los inconvenientes que presenta el balanceo de carga estático se tienen en cuenta. Esto es posible porque la división de la carga computacional depende de las tareas que se están ejecutando y no de la estimación del tiempo que pueden tardar en ejecutarse. Aunque el balanceo de carga dinámico lleva consigo una cierta sobrecarga durante la ejecución del programa, resulta una alternativa mucho más eficiente que el balanceo de carga estático.

En el balanceo de carga dinámico, las tareas se reparten entre los procesadores durante la ejecución del programa. Dependiendo de dónde y cómo se almacenen y repartan las tareas el balanceo de carga dinámico se divide en:

- Balanceo de carga dinámico centralizado. Se corresponde con la estructura típica de Maestro/Esclavo.
- Balanceo de carga dinámico distribuido o descentralizado. Se utilizan varios maestros y cada uno controla a un grupo de esclavos.

  Balanceo de carga dinámico centralizado: El nodo maestro es el que tiene la colección completa de tareas a realizar. Las tareas son enviadas a los nodos esclavos. Cuando un nodo esclavo finaliza una tarea, solicita una nueva al maestro. Esta técnica también se denomina programación por demanda o bolsa de trabajo, y no sólo es aplicable a problemas que tengan tareas de un mismo tamaño.

  En problemas con tareas de distintos tamaños es mejor repartir primero aquellas que tengan una mayor carga computacional. Si la tarea más compleja se dejase para el final, las tareas más pequeñas serían realizadas por esclavos que después estarían esperando sin hacer nada hasta que alguno completara la tarea más compleja. También se puede utilizar esta técnica para problemas donde el número de tareas pueda variar durante la ejecución.

  En algunas aplicaciones, especialmente en algoritmos de búsqueda, la ejecución de una tarea puede generar nuevas tareas, aunque al final el número de tareas se debe de reducir a cero para alcanzar la finalización del programa. En este contexto se puede utilizar una cola para mantener las tareas pendientes. Si todas las tareas son del mismo tamaño y de la misma importancia o prioridad, una cola FIFO (First In First Out) puede ser más que suficiente, en otro caso debe analizarse la estructura más adecuada [15].

  Balanceo de carga dinámico distribuido o descentralizado: Una gran desventaja del balanceo de carga dinámico centralizado es que el nodo maestro únicamente puede repartir una tarea cada vez, y después de que haya enviado las tareas iniciales sólo podrá responder a nuevas peticiones de una en una. Por tanto, se pueden producir colisiones si varios esclavos solicitan peticiones de tareas de manera simultánea.

  La estructura centralizada únicamente será recomendable si no hay muchos esclavos y las tareas son intensivas

computacionalmente. Para tareas de grano fino (tareas con muy poca carga computacional) y muchos esclavos es apropiado distribuir las tareas en más de un sitio [15].

C. *Trabajos Previos Sobre Balanceo de Carga*

Teniendo en cuenta que el balanceo de carga es el esfuerzo por mantener todos los procesadores del cluster realizando algún trabajo productivo [16], a continuación se hará una revisión de algunos trabajos realizados en esta problemática y su aporte a la sociedad.

En 1997 Decker presenta la herramienta VDS, que distribuye automáticamente los paquetes generados por aplicaciones paralelas y balancea la carga entre los nodos de una red [17].

Bohn y Lamont en 1999 investigan técnicas para realizar el balanceo de carga asimétrico en un cluster heterogéneo, mostrando que cuando la diferencia de poder de procesamiento entre los nodos de un cluster es pequeña los beneficios alcanzados con el balanceo asimétrico son pequeños, en caso contrario, se obtiene ganancia de desempeño evitando los procesadores más lentos [18]. En este mismo año Bevilacqua, trabaja con cluster heterogéneos. Él propone un método eficiente de balanceo de carga en un cluster de estaciones de trabajo. Básicamente un nodo maestro es el responsable de enviar la carga a cada nodo ocioso una vez que cada uno lo solicite. Cuando el maestro no esté atendiendo los pedidos, este trabaja los datos que almacena. Los resultados experimentales alcanzan una eficiencia superior al 90% [19].

En el año 2000, Aversa y Bestavros proponen validar la implementación de un prototipo de servidor Web distribuido con el fin de sacar el mejor provecho de balanceo de carga. En este trabajo se obtienen resultados positivos en cuanto a escalabilidad y costo para aplicaciones pequeñas [20].

Para el 2002, el principal objetivo de Kacer y Tvrdík fue el de verificar el adecuado balanceo de carga para procesos cortos usando intensamente el procesador. Como resultado de este estudio se propuso una técnica de ejecución remota de procesos, la cual fue comparada con la estrategia de balanceo adoptada en Mosix. Con las pruebas realizadas, los resultados apuntaron que la técnica propuesta fue superior a la migración de procesos utilizada en Mosix en muchos de los casos y presenta resultados semejantes en los demás [21].

Ibrahim y Xinda evalúan el desempeño de sistemas paralelos que usan balanceo de carga dinámico en términos de tiempo de ejecución, velocidad y eficiencia en ejecución de una versión paralela de un algoritmo de búsqueda de profundidades (depth-first), sobre la plataforma MPI [22].

En 2003 Choi y sus compañeros proponen una métrica de carga basada en número de tareas las cuales afectan el desempeño del sistema. Los resultados de su trabajo muestran una ganancia de un 11% en el tiempo de ejecución [23]. Chau y Fu proponen otra técnica de balanceo de carga que se basa en arquitecturas hipercubo con el objetivo de reducir el tráfico de mensajes [24]. Y en este mismo año DrozDowski y Wolniewicz proponen el modelo de carga divisible, el cual puede ser utilizado cuando las aplicaciones permiten dividir la aplicación en partes de tamaño arbitrario. La principal contribución de este trabajo fue la modelación matemática del problema de carga divisible utilizando jerarquías de memoria [25].

Attiya y Hamam con el objetivo de reducir el tiempo de ejecución del propio algoritmo de distribución de carga, proponen en 2004 un algoritmo ejecutado en dos fases, cada una de estas fases está basada en una heurística diferente: Simulated Annealing y Branch-and-Bound [26].

En 2004 Savvas y Kechadi propusieron un algoritmo llamado PSTS, en el cual la idea principal era dividir recursivamente el cluster en subespacios y encontrar la dimensión que suministrara el mejor desempeño. La propuesta se mostró eficiente para sistemas que permanecen desbalanceados por largo tiempo [27].

Para el año 2006 Rego propone el uso de algoritmos de balanceo de carga sobre la plataforma LAM/MPI. La validación de su trabajo se hace sobre aplicaciones diferentes, como multiplicaciones matriciales y reconocimiento de secuencias de DNA [3].

IV. MÉTODO PARA EL MANEJO DEL BALANCEO DE CARGA

En este trabajo se propondrá un método para balancear la carga de trabajo tratando de optimizar al máximo el tiempo total de ejecución. Considerando esto y en base a lo descrito en el numeral 2, el tipo de cluster seleccionado es el de alto rendimiento (HP, high performance), ya que cuando la palabra cluster es pronunciada, la primera cosa que pasa por la cabeza es alto desempeño. De acuerdo a [3], este tipo de cluster es el más común entre las comunidades científicas, sistemas predictivos, simulaciones y tarifas típicas que exigen alto poder de procesamiento. Su función es sencilla: dado un problema complejo e identificado como paralelizable, un servidor (maestro) debe ser responsable de dividir este problema en numerosas partes para ser procesadas en nodos esclavos (nodos dedicados al procesamiento). Así, una vez que cada nodo esclavo encuentre una solución, este la envía al nodo maestro para que el maestro presente la solución completa del problema.

Se trabajará con el balanceo de carga dinámico centralizado, el cual según [15], consiste en que la asignación de la carga es efectuada por el nodo maestro a todos sus esclavos.

El nodo maestro será el encargado de recibir todas las tareas que se deberán ejecutar y tomar la mejor decisión al momento de realizar la repartición de éstas teniendo siempre en cuenta cuál nodo es más o menos fuerte dentro del cluster. En el método se verá que el nodo maestro no realizará ninguna de las tareas, sólo las distribuirá. De ahí, se descarta todo lo que concierne con el balanceo de carga estático, donde el nodo maestro

participa activamente de la ejecución de las tareas en la computación una vez que haya asignado el trabajo a los nodos esclavos realizando este proceso una vez o cíclicamente.

A. Descripción del Método

El método recopila ideas de dos trabajos revisados previamente en el numeral 3. El trabajo realizado por DrozDowski y Wolniewicz en el que se propone un modelo de carga divisible, el cual se utiliza sólo en aplicaciones que permitan dividirla en tamaños diferentes [25], y el trabajo de Choi, Yu, Kim y otros, en el que el desempeño del sistema tiene mucho que ver con la cantidad de tareas para cada nodo [23].

En este trabajo el método que se propone consiste en la distribución de tareas en los nodos esclavos por parte del nodo maestro, el cual recopilará una a una las tareas conforme van llegando a medida que pasa el tiempo. La asignación de las tareas a los nodos esclavos dependerá del tamaño de las tareas y de una función de aptitud, la cual es calculada por el nodo maestro después de que el nodo esclavo envía ciertos atributos (velocidad de procesamiento, capacidad de memoria y tiempo de respuesta). Después de que el nodo maestro asigna una tarea a cierto nodo esclavo, el esclavo ejecutará la tarea y enviará al nodo maestro una notificación de terminación de tarea, indicándole así al nodo maestro que esta listo para ejecutar una nueva.

La función de aptitud se define como la rapidez asociada a la velocidad de procesamiento, la capacidad de memoria y el tiempo de respuesta o latencia.

El cálculo de la función de aptitud corresponde a la sumatoria entre los atributos mencionados previamente, donde cada uno de estos estará multiplicado por un peso que dependerá de la importancia de este factor en el cluster.

$$FuncionAptitud_{(i)} = cp_{(i)} * a + cm_{(i)} * b + tr_{(i)} * c \quad (1)$$

donde:
cp(i) = velocidad de procesamiento del nodo i
cm(i) = capacidad de memoria del nodo i
tr(i) = tiempo de respuesta o latencia del nodo i
a, b, c = pesos asociados a cada variable y que dependen del tipo de cluster seleccionado

Para hacer el cálculo de la función de aptitud en los nodos es necesario normalizar cada una de las variables, ya que si se trabaja con los valores originales la escala numérica sería muy diferente y la función de aptitud se tornaría altamente variada para los diversos nodos.

En este método se tendrán en cuenta los tamaños de las tareas para la asignación, recopilando estos por medio de series de tandas que se crearan acorde a la cantidad de nodos esclavos.

1. Se toman dos vectores iniciales: en un vector se almacenan todas las tareas iniciales organizadas de mayor a menor tamaño cada una de estas presenta un ID que identifica que tarea es; el otro vector recopila los nodos esclavos organizados por función de aptitud de mayor a menor.

2. Se calculan los valores de las sumatorias de los tamaños de las tareas (STT) y de las funciones de aptitud de los nodos esclavos (SFA).

$$STT = \sum_{i=1}^{n} TamañoTarea_{(i)} \quad (2)$$

$$SFA = \sum_{i=1}^{m} FuncionAptitud_{(i)} \quad (3)$$

Donde n es la cantidad total de tareas que se presentan inicialmente y m es la cantidad de nodos del cluster. El nodo 0 es el nodo maestro.

3. Se calcula el porcentaje de carga (FactorNodo(i)) que podrá recibir cada uno de los nodos esclavos como sigue a continuación:

$$FactorNodo_{(i)} = \frac{fa_{(i)} * 100}{SFA} \quad (4)$$

Donde FactorNodo(i) corresponde al porcentaje de la carga de trabajo o tareas del nodo (i) relacionada con su propia capacidad.

4. Se calcula la capacidad que posee cada nodo con respecto a los tamaños de las tareas, con el fin de obtener un rango de tamaños propios para cada nodo de acuerdo a las tareas existentes hasta el momento.

$$CentroIntervaloNodo_{(i)} = STT * FactorNodo_{(i)}$$
$$RangoNodo_{(i)} = CentroIntervaloNodo_{(i)} \pm error \quad (5)$$

El error se precisó en un 10%, lo que equivale a un manejo de eficiencia del 90%.

$$LimiteInferiorNodo_{(i)} = CentroIntervaloNodo_{(i)} - error$$
$$LimiteSuperiorNodo_{(i)} = CentroIntervaloNodo_{(i)} + error \quad (6)$$

5. En este punto, el maestro ya conoce cual es la capacidad de cada nodo esclavo y procede con la distribución de las tareas teniendo en cuenta los cálculos realizados hasta el paso 4 y que las tareas se encuentran organizadas de mayor a menor tamaño y los nodos se encuentran también organizados de mayor a menor pero en cuanto a función de aptitud.

Para iniciar con la asignación se deberá recorrer el vector en donde se encuentran almacenadas las tareas. Para cada nodo esclavo se creará un nuevo vector llamado Tanda(i) que contendrá las tareas que le serán asignadas para ejecutar.

En el nodo maestro se almacenará una variable para cada uno de los nodos esclavos que será un contador (sumatoria de tamaños de tareas) que examinará si la tarea a ser asignada podrá o no estar en la tanda de tareas del nodo respectivo

cumpliendo o no con los requisitos del rango de cada nodo esclavo. Esta variable se conocerá como SumatoriaTanda(i).

A continuación se realiza la asignación de las tareas a los nodos teniendo en cuenta la variable SumatoriaTanda(i) almacenada en el nodo maestro para cada nodo esclavo.

a) Se toma el primer tamaño de tarea que se encuentra en el vector de tareas y se acumula en SumatoriaTanda(i).

$$SumatoriaTanda_{(i)} = SumatoriaTanda_{(i)} + TamañoTarea_{(i)} \quad (7)$$

b) Se examina sí SumatoriaTanda(i) es menor que LimiteInferiorNodo(i). De ser cierto, se asigna esa tarea a la Tanda(i), se elimina esta tarea del vector y se evalúa sí hay aun hay tareas en el vector de tareas, si es así se continua con la siguiente tarea del vector hasta que SumatoriaTanda(i) sea menor que LimiteSuperiorNodo(i), esto implica que la tanda se encuentra lista para ser asignada al nodo y se encuentra dentro del rango admisible por ese nodo.

Se examina si hay más tareas por asignar en el vector de tareas y se continúa con el siguiente nodo realizando el mismo procedimiento.

Sin embargo puede haber casos excepcionales en el que ya se hayan asignado algunas tareas a la tanda y la tarea siguiente a ser asignada sobrepase el LimiteSuperiorNodo(i), en este caso esta tarea no se asigna a dicho nodo y se evalúa para el nodo siguiente, de ahí se puede observar que este nodo aún posee espacio disponible para ejecutar alguna tarea. También puede existir la posibilidad de que la primera tarea del vector de tareas exceda inmediatamente el LimiteSuperiorNodo(i), cuando esto ocurre, se le asigna dicha tarea a este nodo, y se cambia de nodo y tarea.

c) Después de haber asignado las tareas a los nodos esclavos en sus respectivas tandas se debe analizar nuevamente sí aún quedan tareas en el vector de tareas, de ser el caso, estas deberán ser incluidas en una nueva tanda que será destinada a las tareas faltantes y que deberán ser asignadas a las tandas de los nodos esclavos que tengan aún espacio sin exceder en el LimiteSuperiorNodo(i).

d) Sí llegasen a faltar tareas por ser asignadas después de realizar el paso anterior y ninguna entra en el rango de los nodos esclavos entrará una nueva variable a tomar parte de la asignación. Esta será el tiempo de ejecución, que será el tiempo que tarda cada nodo en ejecutar una tarea respectiva. Esta variable es calculada como el espacio (tamaño de la tarea que se esta realizando) sobre la velocidad (función de aptitud del nodo ejecutor de la tarea).

$$TiempoEjecucion_{(i)} = \frac{TamañoTarea_{(i)}}{FuncionAptitud_{(i)}} \quad (8)$$

El procedimiento a seguir es calcular el tiempo de ejecución total acumulado en cada nodo esclavo (TiempoEjecucionTotal(i)), tomando la tanda de tareas de cada nodo (Tanda(i)) y calculando el tiempo de ejecución para cada una de las tareas que tiene dicho nodo.

Posteriormente se procede a tomar la primera tarea que se encuentra en la tanda de tareas faltantes y a calcular un tiempo de ejecución supuesto (TiempoEjecucionSupuesto(i)), para cada nodo con dicha tarea y a acumularlo en el TiempoEjecucionTotal(i) de cada nodo. Luego se procederá con la revisión de cual nodo posee el menor tiempo de ejecución total supuesto (TiempoEjecucionTotalSupuesto(i)), aquel que tenga dicho tiempo será el nodo al cual se le asignará en su tanda esta tarea.

$$TiempoEjecucionSupuesto_{(i)} = \frac{TamañoTareaFALTANTE_{(i)}}{FuncionAptitud_{(i)}} \quad (9)$$

$$TiempoEjecucionTotalSupuesto_{(i)} = TiempoEjecucionTotal_{(i)} + TiempoEjecucionSupuesto_{(i)} \quad (10)$$

Esto se hace hasta que no queden tareas en la tanda de tareas faltantes.

e) Finalmente el nodo maestro envía a sus nodos esclavos una por una las tareas respectivas de las tandas que les corresponden para que comiencen a ejecutarlas. Cada vez que un esclavo finalice una tarea propia este envía una notificación de terminación de tarea con el fin de que el maestro envíe la siguiente tarea de su tanda, haciendo esto hasta que culmine de ejecutar todas sus tareas.

V. ANÁLISIS DE RESULTADOS

El método propuesto se validará teniendo en cuenta el balanceo de carga dinámico centralizado con el caso más genérico existente, en el que la asignación de las tareas por parte del nodo maestro a los nodos esclavos se hace de manera aleatoria como el método que propone [19] en su trabajo. Una vez que cada esclavo ejecuta la tarea que le correspondió enviará una confirmación de terminación de tarea para que el maestro asigne una nueva tarea, hasta efectuar todas las listadas en el arreglo de tareas. Por otro lado también se validará el método propuesto con una modificación del método de Choi, en el que se propone una métrica de carga basada en número de tareas afectando así el desempeño del sistema en cuanto a tiempo de ejecución. [23]. En esta investigación este método se denominará método uniforme.

Los tres (3) métodos fueron programados en C++, integrados al simulador Network Simulator 2 (NS-2), sin embargo existen algunas variaciones en los métodos de Bevilacqua y de Choi. El de Bevilacqua es denominado en este trabajo método aleatorio y varía en que el nodo maestro nunca realiza trabajo de ejecución, simplemente se encarga de asignar las tareas (que se encuentran desorganizadas por tamaño), y en el trabajo de Choi y compañía,

el método como se mencionó anteriormente es denominado método uniforme y se encarga de distribuir uniformemente las tareas en cuanto a tamaños, a los nodos esclavos sin tener muy en cuenta la capacidad que tenga cada uno de estos sólo que todos los nodos posean una carga similar. En este método el vector de tareas se encuentra organizado por tamaños de estas tal y como el método propuesto.

Para ver mayor información sobre el Network Simulator (NS-2) se puede dirigir a la página siguiente: http://www.isi.edu/nsnam/ns/

Para la realización de las pruebas se tomó un cluster heterogéneo a nivel de hardware modificando la cantidad de nodos esclavos y la cantidad de tareas iniciales. Los resultados obtenidos fueron de tomar 3, 5, 10, 20 y 50 nodos esclavos y ejecutando para cada uno de estos casos 5, 10, 20, 50, 100, 200, 500 y 1000 tareas iniciales. Hay que hacer constar que estas tareas sólo hacen parte de las condiciones iniciales y no de la cantidad total de tareas que ejecuta el cluster como tal, ya que el nodo maestro cuenta con un temporizador capaz de generar más tareas.

Los valores que se trabajaron en la simulación para los pesos asociados en la función de aptitud para cada uno de los nodos esclavos fueron los siguientes:

$a = 0.5; b = 0.35; c = 0.15$

Estos valores fueron escogidos de esta forma debido a que el tipo de cluster utilizado es de alto rendimiento, y como tal se requiere mayor velocidad de procesamiento, seguido de la capacidad de memoria y el tiempo de latencia.

Si se quisiera un cluster de otro tipo (disponibilidad o confiabilidad), sería necesario cambiar los pesos acorde al tipo a la preferencia del usuario; esto es, si fuese cluster de alta disponibilidad sería necesario utilizar mayor peso en la capacidad de memoria, seguido del tiempo de latencia y un peso menor para la velocidad de procesamiento. Pero si se quisiera un cluster de alta confiabilidad los valores cambiarían también.

Cabe aclarar que el algoritmo funciona para cualquier modificación en los pesos, sólo alterando el orden en el que se encuentren los nodos (mejor a peor aptitud) y no a la toma de decisión acorde al método como tal.

Las siguientes tablas y gráficas presentan los tiempos de ejecución totales del algoritmo con 3, 10 y 50 nodos esclavos y con cada una de las condiciones iniciales mencionadas anteriormente acerca de la cantidad de tareas iniciales, sin embargo la cantidad total de tareas ejecutadas dentro del cluster es la suma entre tareas iniciales y tareas generadas por el nodo maestro.

TABLA I. RESULTADOS DE LA EJECUCIÓN DE LA CARGA DE TRABAJO CON 3 NODOS ESCLAVOS

Cantidad de tareas totales ejecutadas	Tiempo de ejecución para cada método [seg]		
	Aleatorio	Uniforme	Propuesto
99	6,361607	6,449856	5,71228
141	8,585431	8,912146	7,595445
149	8,767952	9,413576	8,042756
250	14,42972	15,71695	13,50117
396	23,14886	24,54513	21,01382
479	27,21146	29,05732	25,03099
553	31,74227	34,24748	29,50111
1179	66,19169	71,70027	61,83991

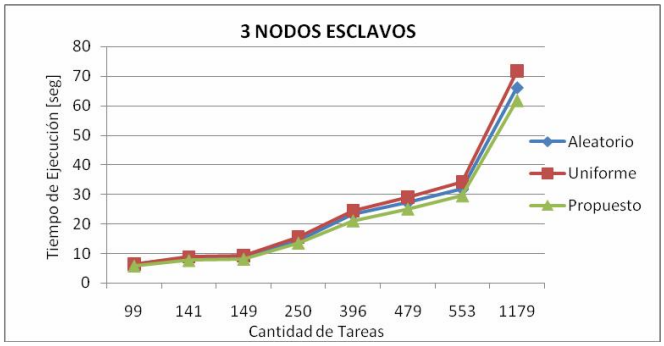

Figura 1. Resultados de la ejecución de carga de trabajo con 3 nodos esclavos.

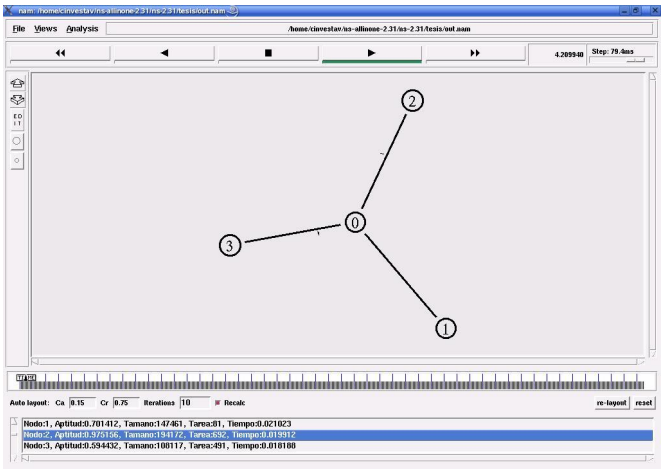

Figura 2. Simulación en el Network Simulator 2 (NS-2) con 3 nodos esclavos.

TABLA II. Resultados de la Ejecución de la Carga de Trabajo con 10 Nodos Esclavos

Cantidad de tareas totales ejecutadas	Tiempo de ejecución para cada método [seg]		
	Aleatorio	Uniforme	Propuesto
46	1,981356	2,010801	1,210137
121	4,891438	6,420832	2,871366
222	6,575336	11,58503	5,290643
273	9,287934	14,64966	6,429258
365	11,01577	18,82134	8,330664
496	15,14604	26,10157	11,38643
832	26,34462	43,12187	18,79545
1209	36,80888	62,41019	27,19923

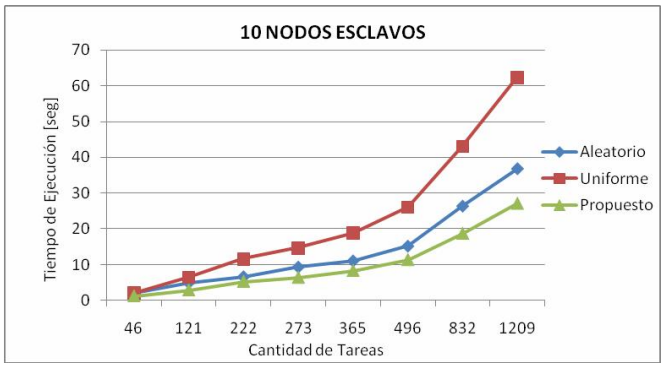

Figura 3. Resultados de la ejecución de carga de trabajo con 10 nodos esclavos.

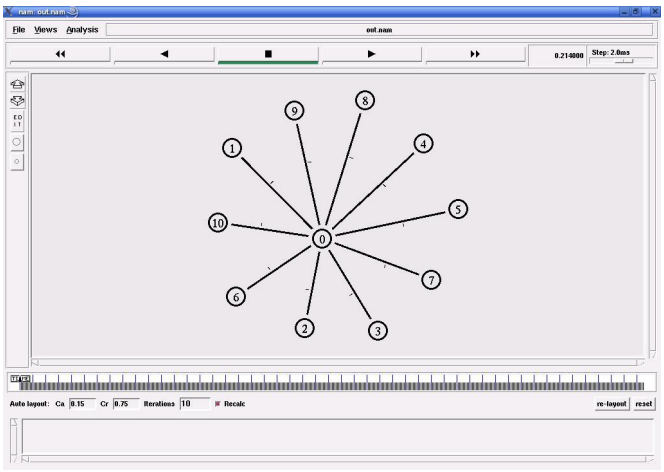

Figura 4. Inicio de la simulación en el NS-2 con 10 nodos esclavos.

De los resultados se puede observar que para todos los casos el método propuesto mejora los tiempos de ejecución, sin embargo se puede apreciar que los mejores resultados ocurren cuando se trabaja con mayor cantidad de tareas y mayor cantidad de nodos, por lo que se podría concluir que el método es altamente escalable y minimiza el tiempo total de ejecución cumpliendo muchas de las expectativas de esta investigación.

TABLA III. RESULTADOS DE LA EJECUCIÓN DE LA CARGA DE TRABAJO CON 50 NODOS ESCLAVOS

Cantidad de tareas totales ejecutadas	Tiempo de ejecución para cada método [seg]		
	Aleatorio	Uniforme	Propuesto
100	1,324659	0,784898	0,7829418
198	1,844983	1,995441	1,179421
218	2,034206	2,474166	1,325206
318	2,713001	3,965946	1,483486
387	3,428265	4,141913	1,762839
387	3,492864	4,120583	1,778959
785	5,918553	9,403513	3,50783
1166	7,719521	13,71055	5,286702

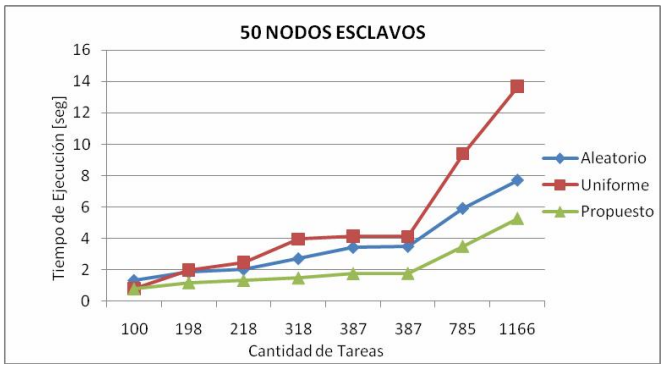

Figura 5. Resultados de la ejecución de carga de trabajo con 50 nodos esclavos.

La forma en cómo funcionan cada uno de los métodos en cuanto a la distribución de las tareas puede verse en las ilustraciones siguientes en donde para cada uno de los métodos la distribución es diferente y el tiempo de ejecución también se ve afectado tal y como se vio en los resultados experimentados. Se verán los resultados con 10 nodos esclavos y 10 tareas iniciales, de las cuales al final se generan dos tandas más de tareas una 165 y otra de 190 haciendo un acumulado total de 365 tareas totales.

TABLA IV. FUNCIONES DE APTITUD DE 10 NODOS ESCLAVOS

Nodo	Función deAptitud
2	0,875157
6	0,782739
1	0,641703
4	0,54459
3	0,542113
10	0,514381
7	0,439945
5	0,39396
9	0,360987
8	0,210635

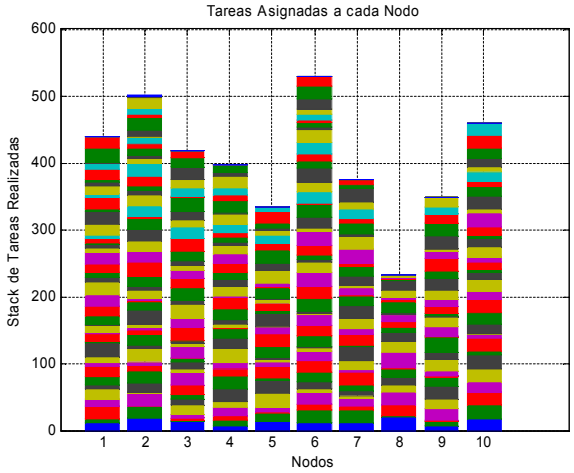

Figura 6. Método aleatorio con 10 nodos esclavos para 10 Tareas Iniciales como condición inicial.

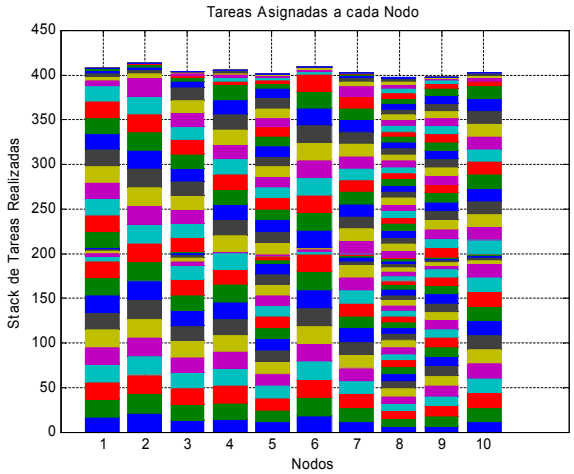

Figura 7. Método uniforme con 10 nodos esclavos para 10 Tareas Iniciales como condición inicial.

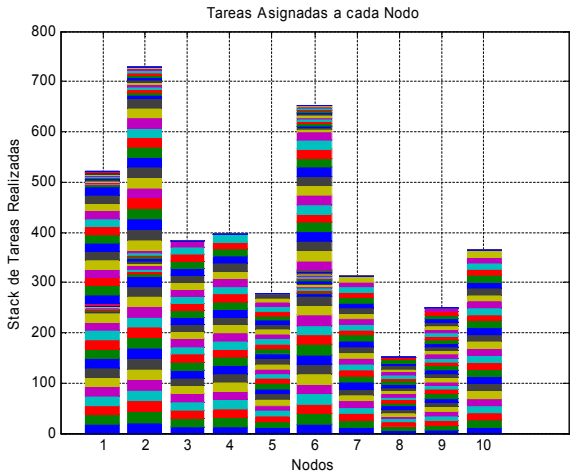

Figura 8. Método propuesto con 10 nodos esclavos para 10 Tareas Iniciales como condición inicial.

De la tabla 4 y la figura 8 se puede apreciar que el nodo con mayor eficiencia por su velocidad es el nodo 2 y el de menor aptitud es el nodo 8 en el método propuesto corroborando que la asignación de carga se hace en base a la capacidad de cada nodo.

Por otro lado, en la figura 7 se observa que la asignación en el método uniforme en cuanto a carga es similar para cada nodo esclavo, por lo que el tiempo de ejecución total en este caso es determinado por el nodo más lento, en este caso por la función de aptitud asociada es el nodo 8, dato que es fácilmente observado en la tabla 2 y figura 3 donde el tiempo de ejecución para 356 tareas es de 18,82134 segundos.

Para el método aleatorio se aprecia que al momento de hacer peticiones de tareas y asignarlas por parte del maestro, sólo se hace si un nodo culminó su tarea actual y en la pila de tareas pendientes aún quedan por asignar. De la figura 6 se observa que el nodo con menor asignación de carga es el nodo 8, y esto se debe a que su velocidad (función de aptitud = 0,210635) no alcanza para cubrir con mayor capacidad de carga y por esto no puede pedir gran cantidad de tareas.

VI. CONCLUSIONES

Debido al rápido avance de la tecnología y a la necesidad de utilizar los procesadores de los equipos existentes, las diferentes comunidades científicas se vieron en la tarea de utilizar cluster heterogéneos, sin embargo conectar cada uno de los nodos con recursos de tipos y/o capacidades distintas puede ayudar al bajo desempeño sí las diferencias en el poder de procesamiento entre los nodos no son observadas y tratadas de forma correcta.

Un cluster destinado a aplicaciones paralelas de alto desempeño y formado por nodos heterogéneos puede presentar bajo desempeño si todos los nodos son tratados indiscriminadamente. Este bajo desempeño tiende a ser más expresivo cuanto mayor sea la diferencia entre el poder computacional de los nodos. Buscando aumentar el desempeño en el cluster son aplicadas algunas técnicas de balanceo de carga, en las que se hace que los nodos reciban carga de acuerdo a sus propias capacidades en cuanto a recurso computacional se refiere.

En este trabajo se enfatizó en que cada nodo sólo pudiera efectuar trabajos que no excedieran sus capacidades reales con el fin de optimizar al máximo los recursos del cluster adoptando cargas grandes sólo por procesadores capaces. Así, el tiempo total de ejecución se logró minimizar en grandes porciones por el hecho de que cada nodo efectuaba lo que podía realizar sin exceder un límite.

VII. TRABAJO FUTURO

Como trabajo futuro se propone escalar este método a sistemas reales con el fin de que éste pueda ser utilizado en grandes industrias y en tiempo real.

Por otro lado también se propone realizar algún trabajo en el que se estudie el cómo hacer que la función de aptitud sea variable, implementando por ejemplo, un sistema experto basado en reglas con el fin de que no sólo sea un algoritmo de balanceo de carga enfatizado en rendimiento sino también en confiabilidad y disponibilidad, ó simplemente un algoritmo genérico basado en los requerimientos del usuario final.

AGRADECIMIENTOS

A todas aquellas personas interesadas en el tema y que por sus comentarios y sugerencias aportaron para que la culminación de este artículo fuera un hecho. Y en especial a muchos de los estudiantes e investigadores del Centro de Investigación y Estudios Avanzados (Cinvestav) del IPN Unidad Guadalajara, porque sin su ayuda el inicio de este trabajo no hubiese sido posible.

REFERENCIAS

[1]E.J. Plaza. "Cluster heterogéneo de computadoras". 2004. DOI= http://www.ii.uam.es/~fjgomez/CursoVerano/instal/cluster.pdf.

[2]L.F. Cueto. Estudio e implementación de un cluster clase beowulf.

[3]F.V. Rego. "Balanceamento de carga em clusters de alto desempenho: uma extensão para a LAM/MPI". Universidade Estadual de Maringá. 2006.

[4]R. Chirinov. "Proyecto Cluster openMosix (Linux)." 2003. DOI= http://www.noticias3d.com/articulo.asp?idarticulo=248&pag=4

[5]O. Pino, R.F. Arroyo y F.J. Nievas. "Los Clusters como Plataforma de Procesamiento Paralelo". 2002. DOI= http://usuarios.lycos.es/lacaraoculta/descargas/Clusters_definitivo.pdf

[6]DOI= http://www.linuxiso.cl/manuales/doc-manual-openMosix-1.0/doc-manual-openMosix_html-1.0/node16_ct.html

[7]C. Lizárraga. "Cluster de Linux". Departamento de Física. Universidad de Sonora. México. 2002. DOI= http://clusters.fisica.uson.mx/

[8]G. Chiola. "Some Research Projects on Clusters of Personal Computers". 24th. Proceedings of Euromicro Conference. Volume 2, pág. XLVII - XLLIV, del 25 al 27 de Agosto de 1998.

[9]AUTOR DESCONOCIDO. "Superordenadores". Salamanca (España). DOI = http://html.rincondelvago.com/superordenadores.html

[10]K.E. Hoganson. "Workload execution strategies and parallel speedup on clustered computers". IEEE Transactions Computers. Volume 48, número 11, pág. 1173 - 1182, noviembre de 1999.

[11]V.A.K. Nguyen. y S. Pierre. "Scalability of computer clusters". Electrical and Computer Engineering. Canadian Conference on Volume 1, pág. 405 - 409, de 13 a 16 de mayo de 2001.

[12]D. Andresen, N. Schopf, E. Bowker, et al. "DISTOP: A low-overhead cluster monitoring system". In Proceedings of the PDPTA. Las Vegas, pág. 1832 - 1836, junio de 2003.

[13]D. Avresky y N. Natchev. "Dynamic reconfiguration in computer clusters with irregular topologies in the presence of multiple node and link failures". IEEE Transactions Computers. Volume 54, número 5, pág. 603 - 615, mayo de 2005.

[14]J. Haase, F. Eschmann y K. Waldschmidt. "The SDVM - An Approach for Future Adaptive Computer Clusters". Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05). Washington, volume 17, pág. 278a - 278a, de 4 a 8 de abril de 2005.

[15]S. Dormido, R. Hernández, S. Ros y J. Sánchez. "Procesamiento Paralelo. Teoría y Programación". Editorial Sanz y Torres. Madrid, Enero de 2003.

[16]W. George. "Dynamic load-balancing for data-parallel MPI programs". In Message Passing Interface Developer's and User's Conference (MPIDC'99). pág. 95 - 100, 1999.

[17]T. Decker. "Virtual Data Space - A universal load balancing scheme". In Proceedings of the 4-th International Symposium on Solving Irregularly Structured Problems in Parallel. Volume 1253 of Lecture Notes in Computer Science, pág. 159 - 166, 1997.

[18]C.A. Bohn y G.B. Lamont. "Asymmetric load balancing on a heterogeneous cluster of pcs". In Proceedings of the PDPTA. Las Vegas, volume 5, pág. 2515 - 2522, 1999.

[19]A. Bevilacqua. "Dynamic load balancing method on a heterogeneous cluster of workstations". Informatica. Volume 23, número 1, pág. 49 - 56, marzo de 1999.

[20]L. Aversa y A. Bestavros. "Load balancing a cluster of web servers using distributed packet rewriting". IEEE Int'l Performance, Computing and Communication Conf. pág. 24 - 29, 2000.

[21]M. Kacer y P. Tvrdík. "Load balancing by remote execution of short processes on linux clusters". IEEE/ACM International Symposium on Cluster Computing and the Grid, 2002.

[22]M.A. Ibrahim y L. Xinda. "Performance of dynamic load balancing algorithm on cluster of workstations and PCs". Proceedings of Algorithms and Architectures for Parallel Processing. Fifth International Conference on 23-25, pág. 44 - 47, octubre de 2002.

[23]M. Choi, J. Yu, H. Kim, et al. "Improving performance of a dynamic load balancing system by using number of effective tasks on Cluster Computing". Proceedings of IEEE International Conference. pág. 436 - 441, 2003.

[24]S.C. Chau y A.W. Fu. "Load balancing between computing clusters". Proceedings of the Fourth International Conference. Parallel and Distributed Computing, Applications and Technologies, PDCAT'2003. pág. 548 - 551, 27-29. Agosto de 2003.

[25]M. Drozdowski y P. Wolniewicz. "Out-of-core divisible load processing". IEEE Transactions on Parallel and Distributed Systems, 14, 10, pág. 1048 - 1056, 2003.

[26]G. Attiya y Y. Hamam. "Two phase algorithm for load balancing in heterogeneous distributed systems". Proceedings of 12th Euromicro Conference Parallel, Distributed and Network-Based Processing. on 11-13, pág. 434 - 439, Febrero de 2004.

[27]I.K. Savvas y M.T. Kechadi. "Dynamic task scheduling in computing cluster environments". Parallel and Distributed Computing. Third International Symposiumon/Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, 2004. Third International Workshop, pág. 372 - 379, 5 a 7 de julio de 2004.

Félix Francisco Ramos Corchado. Ingeniero en Electrónica en Comunicaciones de la Universidad Autónoma Metropolitana Iztapalapa. MSc en Ciencias con Especialidad en Computación del Cinvestav Ciudad de México. Diploma de Estudios Avanzados con Especialidad en Sistemas Distribuidos del Instituto Nacional de Telecomunicaciones en París, Francia. Ph.D en Computación de la Universidad de Tecnología de Compiègne en Francia. Miembro del SIN (Nivel 1). Áreas de interés: Algorítmica Distribuida, Sistemas Multiagentes, Inteligencia Artificial Distribuida, Realidad Virtual Distribuida y Aumentada, Trabajo Cooperativo Asistido por Computador.

John William Branch Bedoya. Ingeniero de Minas y Metalurgia de la Universidad Nacional de Colombia, Sede Medellín. MSc. en Ingeniería de Sistemas de la Universidad Nacional de Colombia. Ph.D en Ingeniería de Sistemas de la Universidad Nacional de Colombia. Áreas de interés: Procesamiento Digital de Imágenes, Computación gráfica, Realidad Virtual, Reconocimiento de Patrones, Visión 3D, Inteligencia Computacional, Procesamiento Paralelo y Distribuido.

Andrea Mesa Múnera. Ingeniera de Sistemas e Informática de la Universidad Nacional de Colombia, Sede Medellín. Candidata a Magíster en el Postgrado, Maestría en Ingeniería de Sistemas de la Universidad Nacional de Colombia. Áreas de interés: Redes Teleinformáticas, Procesamiento Paralelo y Distribuido, Educción y Análisis de Requisitos de Ingeniería de Software.

Raúl Esteban Jiménez Mejía. Estudiante de Ingeniería Eléctrica de la Universidad Nacional de Colombia, Sede Medellín. IEEE Student Member. Áreas de interés: Redes Teleinformáticas, Procesamiento Paralelo y Distribuido, Sistemas de Potencia, Procesamiento de Imágenes, Identificación de Sistemas Dinámicos.