ANÁLISIS DE TÉCNICAS DE AJUSTE Y DISTRIBUCIÓN DE CARGAS COMPUTACIONALES EN SISTEMAS MULTIAGENTE

Diego Ceñal Álvarez (el_dekano@hotmail.com)
Plaza del Bierzo, Nº6 4ºD, 24010, León, España.

Jose Ramón Villar Flecha (diejvf@unileon.es), Angel Alonso Alvarez (dieaaa@unileon.es), Isaías Garcia (dieigr@unileon.es) y Carmen Benavides (diecbc@unileon.es)
Área de Ingeniería de Sistemas y Automática, Departamento de Ingeniería Eléctrica y Electrónica, Universidad de León, Campus de Vegazana s/n 24071 León. España.

RESUMEN

Este trabajo forma parte de un proyecto global cuyo objetivo es desarrollar un sistema basado en conocimiento para teoría de control. Es un proyecto multidisciplinar, en el que se abarcan temáticas tan dispares como las bases de datos o aspectos propios de la inteligencia artificial. El objetivo de esta parte del trabajo es desarrollar una plataforma multiagente sobre la que pueda correr, de forma eficiente, un sistema basado en conocimiento, permitiendo llevar a cabo una distribución de las tareas a realizar entre varios equipos, facilitando un balanceo de la carga del sistema y, en general, el uso eficiente de los recursos disponibles.

Palabras Clave: Balanceo de Carga, Agentes, OpenMosix, Sistema Distribuidos, Sistemas Multiagente.

1. INTRODUCCIÓN

Los Sistemas Basados en Conocimiento (KBS) son una rama de la Inteligencia Artificial [1] y, básicamente, se componen de una base de conocimiento (KB), que refleja la estructura conceptual del dominio, y una serie de métodos de resolución de problemas, usados para decidir y construir las acciones a realizar [8]. La base de conocimiento se compone de ontologías, que son las especificaciones formales de conceptos y métodos propios del dominio, objetos que representan algún tipo de elemento real y de las relaciones existentes entre todos estos elementos. Toda esta información referente al dominio en cuestión debe estar a disposición del sistema.

El proyecto CEKB (Control Engineering Knowledge Base), combina todos los aspectos de la ingeniería de control y de la ingeniería de conocimiento de forma que el objetivo final es generar un sistema basado en conocimiento sobre ingeniería de control, el cual queda reflejado en la figura 1.

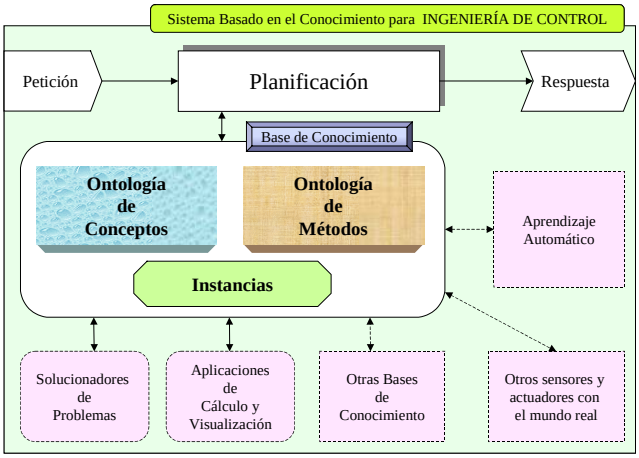

Figura 1 : Estructura de CEKB

Cabe destacar la base de conocimiento que ocupa la posición central y las aplicaciones que hacen uso de la misma están distribuidas alrededor de ella.

El sistema recibirá peticiones de entrada por medio de un modulo que actúa de interfaz. El modulo planificador es el encargado de solucionar la petición utilizando los métodos y conceptos existentes en la base de conocimiento y el resultado será devuelto por otro modulo que actúa de interfaz de salida. Los métodos utilizados para resolver los problemas que encuentran definidos dentro de la base de conocimiento, pero para que estos puedan ejecutarse es necesario disponer de unas librerías externas, que son las únicas que tienen la capacidad de ejecutar dichos métodos. A estas librerías se les llama Solucionadores de Problemas.

Para llevar a cabo algunas de estas operaciones será necesario realizar cálculos complejos, por lo que resulta más apropiado la utilización de aplicaciones especificas para estos cálculos, como por ejemplo MATLAB ©, por lo que el sistema debe contar con algún tipo de interfaz que le permita comunicarse con estas aplicaciones.

Dentro del proyecto CEKB tiene un peso importante una tecnología en auge denominada programación de agentes [2], que aunque no influye en el objetivo final del sistema, facilita su desarrollo y da nuevas características muy interesantes para el funcionamiento general del mismo. El uso de la tecnología de agente tiene el objetivo de implementar la comunicación entre cada uno de los elementos, los cuales se encontraran dispersos, y la posibilidad de realizar balanceo de carga dentro de los equipos utilizados.

El objetivo de esta parte del trabajo es desarrollar una plataforma multiagente [12] sobre la que pueda correr, de forma eficiente, un sistema basado en conocimiento, permitiendo llevar a cabo una distribución de las tareas a realizar entre varios equipos, facilitando un balanceo de la carga del sistema y, en general, el uso eficiente de los recursos disponibles.

En la sección 2 se hablará sobre OpenMosix, plataforma de referencia para nuestro sistema. En la sección 3 se dará una visión del sistema desarrollado. En la sección 4 se mostraran las conclusiones extraídas y futuras líneas de trabajo.

2. OPENMOSIX

Para esta técnica de balanceo de cargas distribuida vamos a utilizar y adaptar las ideas extraídas del proyecto de cluster OpenMosix. La justificación esta en que dicho proyecto no es un mero planteamiento teórico, existen actualmente gran cantidad de clusters comerciales operativos en todo el mundo basados en esta tecnología en empresas tan importantes como Google. Debido a esto queda claro que es una herramienta suficientemente contrastada y, además, de código libre y con una gran cantidad de documentación.

2.1 HISTORIA

El proyecto OpenMosix [11] es un fork libre del proyecto Mosix [9]. El proyecto Mosix fue desarrollado por el profesor Amman Barak y su equipo en la Universidad Hebrea de Jerusalén entre 1977 y 1979 como parte de un proyecto financiado originalmente por el DARPA [3] para crear un cluster de máquinas PDP-11/45, usadas por las fuerzas aéreas de estados unidos, sobre un sistema operativo Unix 6 de Bell Labs.

Debido a los buenos resultados obtenidos, el profesor Barak continuó trabajando en este proyecto, naciendo así Mosix como tal, realizando sucesivas migraciones a lo largo de los años para adaptar el código a los nuevos equipos. En 1998 se decidió migrar a Linux todo el código de Mosix debido a la inminente desaparición de los sistemas operativos utilizados hasta ese momento y para seguir la tendencia natural existente en esa época. Un año después de esta migración, el profesor Barak se vio obligado a liberar el código de Mosix debido a la influencia de la licencia GPL bajo la que fue desarrollado el kernel de Linux.

Esto provocó una gran aceptación por parte de la comunidad Linux y comenzaron a llegar aportaciones y colaboraciones desinteresadas para mejorar o parchear Mosix, pero esto no fue del agrado del profesor Barak, el cual en el año 2002 cambió la licencia de Mosix retirando el derecho de modificación del código de todos los elementos que le fue posible.

En ese momento, un grupo de desarrolladores de Mosix, liderados por Moshe Bar, tomó la última versión de Mosix que se podía demostrar que cumplía la licencia GPL y comenzó el proyecto OpenMosix.

A partir de este momento, OpenMosix tubo un gran impulso, gracias a la colaboración de numerosos programadores que aportaron parches y corrigieron errores en el código, hasta el punto de que OpenMosix se ha convertido en un cluster viable y perfectamente funcional para ámbitos profesionales y manteniendo su licencia GPL.

2.2. QUE ES Y COMO FUNCIONA

OpenMosix [6] es un conjunto de parches del kernel y unas utilidades y bibliotecas de área de usuario que permiten tener un sistema SSI [7] completo para Linux, por lo cual, en pocas palabras, su objetivo es que los procesos puedan migrar dentro de los nodos del cluster de Linux para conseguir un mejor aprovechamiento de los recursos disponibles, siempre de forma transparente al usuario.

Los principales aspectos por los que un proceso migra de un nodo a otro son: por sobrecarga de uso del procesador del nodo de origen, se pasan procesos de las CPU más sobrecargadas a las que tengan menos carga, y por exceso de swap de memoria, teniendo por destino de migración aquellos nodos que tengan menor uso de memoria y, por lo cual, los que minimicen la necesidad de swap para los procesos. Todo este se evalúa teniendo en cuenta el coste de la migración, para evitar migraciones continuas de “Ida y Vuelta” entre dos máquinas.

El modelo de migración de OpenMosix es “Fork and Forget”, una vez se produce la división de un

proceso en dos partes, padre e hijo, estos dos elementos podrán ejecutarse en nodos distintos si así se mejora el aprovechamiento general del cluster, sin que esto suponga un problema de comunicación entre estas dos partes, puesto que padre e hijo seguirán comunicándose de forma local, como si ambos estuviesen en el mismo nodo y es OpenMosix el que se encarga de la gestión de los mensajes cuando ambas partes no están en el mismo nodo.

2.3 MIGRACIÓN

El concepto de migración de procesos consiste en que un proceso se ejecuta en un nodo diferente al que lo generó. Cuando se produce una migración, en el nodo raíz (en el que se creó en proceso) se siguen realizando las llamadas al kernel y aquellas instrucciones que no pueden ejecutarse en este nodo por falta de información o recursos, serán transmitidas al nodo de ejecución del proceso gracias a las modificaciones que los parches de OpenMosix han realizado en el kernel de los equipos.

El algoritmo de equilibrado automático de carga es completamente distribuido, cada nodo decide localmente si va a migrar alguno de los procesos que esta ejecutando y el destino lo decide cuando el nodo que toma la decisión esta sobrecargado basándose en los datos del uso del cluster recolectados por el propio nodo y almacenados localmente. Esto permite crear un cluster de miles de equipos sin que exista un punto critico de fallo o un cuello de botella (punto de sobrecarga) para todo el cluster. En el caso de la caída de un nodo del cluster únicamente se verán afectados los procesos ejecutados y los creados por dicho nodo.

La función de coste del algoritmo se basa en el uso de los recursos del cluster, incluyendo el uso de memoria y procesador entre otros parámetros, e intenta maximizar el uso de los recursos. Es un algoritmo de tipo greedy, por lo cual es muy simple y eficiente, y eso junto con que el algoritmo solo se lanza en un nodo cuando en dicho nodo se ha alcanzado un nivel de carga importante, hace que el calculo de la migración no sobrecargue más aun el nodo. Cada nodo, de forma independiente, realiza preguntas a un grupo de nodos escogidos de forma aleatoria entre todos los nodos del cluster y estos le informan sobre su estado y carga. Con el tiempo, un nodo consigue tener una información de base suficiente para poder tomar decisiones útiles y razonadas en cuanto a la migración.

Como cada nodo solo dispone de información de un subconjunto de nodos del cluster, podría ocurrir que alguna decisión individual no sea la más optima con respecto a todos los nodos disponibles, pero en el aspecto general la toma de decisiones si es optima.

3. APLICACIÓN DESARROLLADA

3.1. INTRODUCCIÓN

El trabajo realizado ha tomado como base las ideas planteadas por el proyecto de cluster OpenMosix, aunque aun no se han implementado todas ellas.

Como acabamos de ver, OpenMosix planteaba un sistema distribuido para el balanceo de carga, de forma que si un equipo fallase en un momento determinado, solo se viesen afectadas aquellas tareas que hayan sido solicitadas desde el dicho nodo y aquellas que, siendo solicitadas por un nodo ajeno, se estuviesen ejecutando en el nodo que ha fallado. Además, este planteamiento buscaba una fácil escalabilidad, de forma que en el momento en que introdujésemos un nuevo equipo formando parte de la red existente, por si solo y después de un tiempo relativamente corto, fuese capaz de formar parte de la red y, por lo cual, capaz también de recibir y enviar solicitudes de ejecución.

Los puntos anteriormente planteados, así como los aspectos referidos a un balanceo de carga transparente en el sistema, han sido alcanzados por nuestra aplicación. A continuación se presentara una pequeña descripción de la plataforma de desarrollo, una explicación de la aplicación y, más adelante, una descripción más exhaustiva de dicha aplicación con el planteamiento original de OpenMosix.

3.2. JADE Y FIPA

Para desarrollar nuestra aplicación utilizaremos una herramienta llamada JADE (Java Agent Developement Framework) [5], la cual esta basada en el estándar FIPA (Fundation for Intelligent Physical Agents) [4].

FIPA es una asociación cuyo objetivo es promocionar el desarrollo de las tecnologías de agentes inteligentes y, para facilitar este proceso, da unas especificaciones acordadas internacionalmente en las que se definen conceptos y estándares para el desarrollo de sistemas multiagente. Se proporciona una arquitectura abstracta para un sistema multiagente, definiendo una plataforma de agentes, los lenguajes que utilizan para comunicarse, los protocolos a seguir para realizar dichas comunicaciones, los elementos necesarios para poder mantener la plataforma en funcionamiento, etc.

Dentro de JADE se incluyen dos productos: una plataforma de agentes (de acuerdo a lo indicado en el estándar FIPA) y un paquete para el desarrollo de agentes en Java. JADE esta codificado en su totalidad en Java, de forma que cualquier desarrollador puede desarrollar el código para sus agentes basándose en las librerías proporcionadas por JADE.

Dentro de una plataforma de agentes tenemos los siguientes componentes:

- AMS (Agent Management System), el agente encargado de supervisar el funcionamiento y el control de acceso a la plataforma. Solo puede asistir un AMS por plataforma. Es el proveedor del servicio de ciclo de vida y de paginas blancas, guardando un listado con todos los identificadores de los agentes que hay en la plataforma junto con la información con respecto a su estado. Los nuevos agentes se registran en el AMS para obtener su identificación, que se llama AID.

- DF (Directory Facilitator), es el agente proveedor del servicio de paginas amarillas de la plataforma, que consiste en mantener un registro de los diferentes servicios que se ofrecen en la plataforma y que agente los ofrece.

- ACC (Agent Comunication Channel) es el agente encargado de controlar el intercambio de mensajes tanto dentro de la propia plataforma como con plataformas remotas.

Estos agentes generalmente trabajan en segundo plano permitiendo a nuestros agentes realizar diferentes operaciones.

3.3. DESCRIPCIÓN DE LA APLICACIÓN

La plataforma desarrollada se basa en tres tipos de agentes, que son:

- Agentes Distribuidores, son los encargados de mantener en funcionamiento la plataforma. Deberá existir uno y solo uno en todos los nodos de nuestra red y tiene dos tareas generales encomendadas:
  - Gestionar la información del entorno, cada uno de los nodos debe almacenar información del estado de carga de otros nodos (si el número de nodos es pequeño, se pregunta a todos, si el número es grande, se pregunta a un subconjunto). Además esta información debe actualizarse de forma periódica, puesto que va a utilizarse para determinar cuales son los nodos más capaces. Este proceso tiene dos partes, por un lado, se deben realizar consultas periódicas al resto de nodos y por otra se deben contestar todas las consultas recibidas de otros nodos.
  - Decidir la ejecución de las tareas, es decir, cuando se quiera ejecutar algo en un nodo, deberá enviar una solicitud al agente distribuidor que se esta ejecutando en ese nodo y será este quien que se encargue de que la tarea solicitada se ejecute en el mejor lugar posible (que puede ser el nodo local o cualquier otro nodo de la plataforma), de forma completamente transparente a quien hizo la solicitud.

- Agentes de Tarea, aquellos agentes que “saben hacer cosas”, es decir, los agentes que disponen del código de aquellas tareas que pueden ser requeridas dentro de la plataforma. No existen limites en cuanto al número o tipo de estos agentes en cada una de los nodos, pudiendo no haber ninguno. Cada uno de estos agentes puede ser capaz de ejecutar una o varias tareas, según como haya sido programado. Estos agentes permanecen a la espera de las solicitudes de ejecución de los agentes distribuidores tanto de su nodo como los de otros nodos.

- Agentes de servicio, son los encargados de realizar las solicitudes de ejecución de una determinada tarea. Su objetivo es simplemente solicitar al agente distribuidor de su nodo la ejecución de una tarea y esperar a que el agente distribuidor de conteste. Si la respuesta del agente distribuidor es negativa, el agente de servicio esperará un tiempo predeterminado, y volverá a realizar la solicitud.

Como hemos comentado anteriormente, uno de los agentes tienes que ejecutar tres tareas diferentes y completamente independientes (Realizar las peticiones de información, Contestar a las peticiones recibidas y Distribuir las tareas). Para que esto sea posible se utilizan los Behaviours (comportamientos), ofrecidos por JADE, que siguen una lógica muy similar a la de los hilos de ejecución de Java. De esta forma un mismo agente arranca varios comportamientos, lo que da una sensación de ejecución paralela de dichas

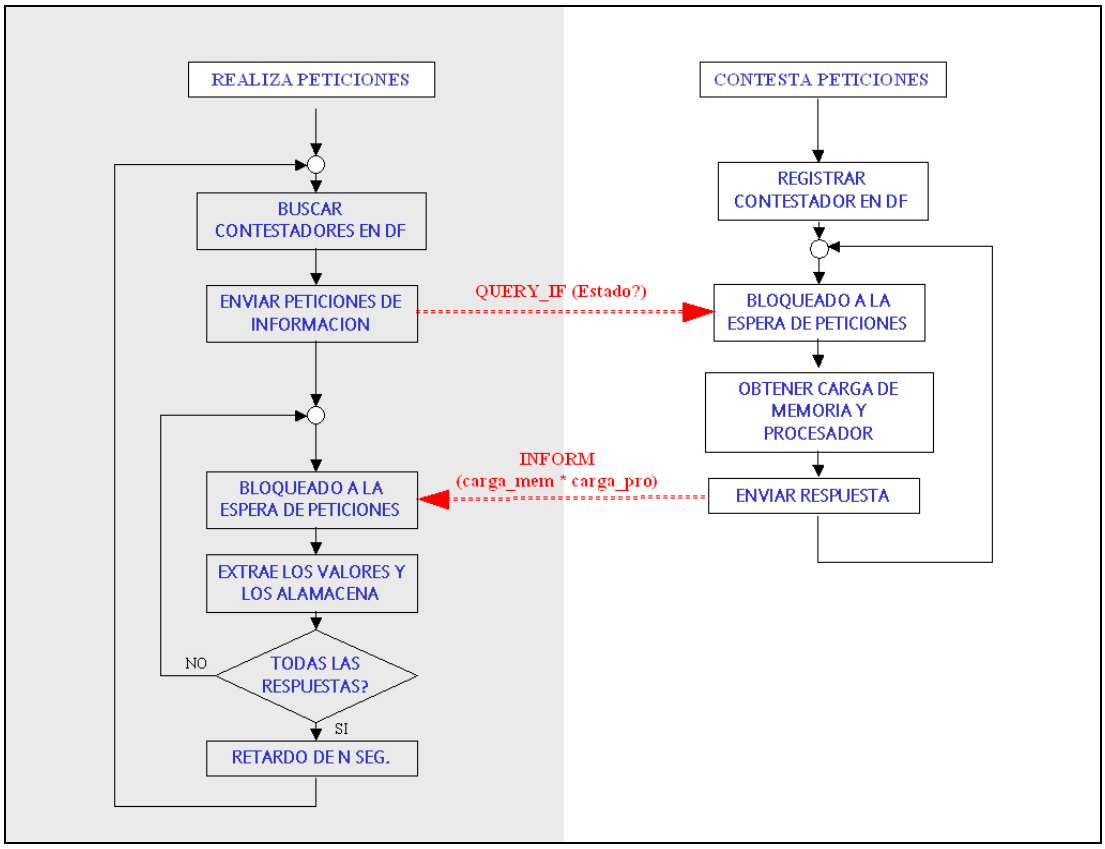

Figura2. Mantenimiento de la información del entorno.

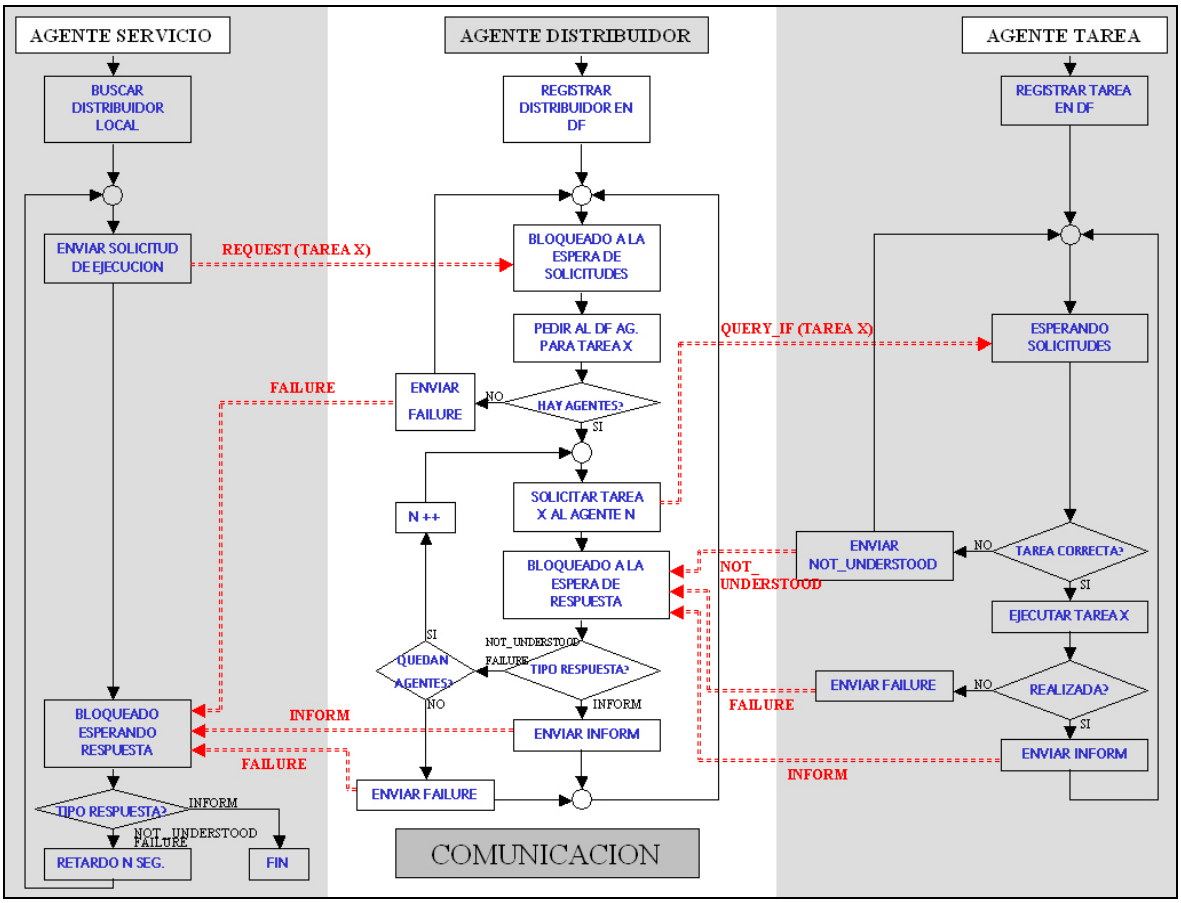

Figura 3: Comunicación completa para la tramitación de una solicitud de ejecución.

aplicaciones. Otra solución podría ser disponer de un agente para cada una de las acciones, de forma que no fuese necesario utilizar los behaviours, pero esta solución no aportaría ventajas de ejecución y, sin embargo, se produciría una sobrecarga de los recursos de la plataforma de agentes. Por lo cual es recomendable el uso de los comportamientos.

En la descripción de los agentes se puede observar que los agentes distribuidores y los agentes de tarea permanecen a la espera de solicitudes, actuando de forma pasiva y reaccionando únicamente cuando un agente de servicio realiza una solicitud, por lo que se podría decir que son los agentes de servicio los que ponen en marcha el sistema.

En la figura 2 se muestra el proceso de comunicación para mantenimiento de información del entorno y en la figura 8 se muestra el proceso de una solicitud de ejecución de una tarea.

En la figura 3 se pueden observar el proceso de comunicación completo, con todas las variantes posibles, en el que un agente de servicio solicita la ejecución de una tarea llamada TAREA X.

4. CONCLUSIONES Y TRABAJOS FUTUROS

El sistema presentado en este trabajo es una primera aproximación a la distribución de la carga computacional con el uso de agentes. Actualmente hay muy pocos trabajos que se centren en este aspecto, y menos aun tal y como se ha planteado esta aplicación, puesto que el objetivo final seria una sistema de agentes que funcionase de forma equivalente a un cluster SSI . Se ha obtenido un sistema operativo y fácil de utilizar y modificar por un desarrollador cualquiera que lo quiera utilizar para ejecutar sus propias tareas. Pero, como ya hemos dicho, esta es una primera fase sobre la que necesitamos realizar futuras modificaciones como son:

- Realizar las modificaciones que aproximen más aun el resultado al funcionamiento de OpenMosix, como son: separar las acciones llevadas a cabo por el Agente Distribuidor de forma que se ejecuten en dos agentes, mejorar los filtros de mensajes. Evaluar las ventajas o desventajas derivadas de modificar la política de ejecución prioritaria, sustituyéndola por otra política en la que solo se tome como referencia la carga del equipo en el que se encuentra el agente, independientemente de que sea local o remoto.

  Con estos primeros cambios se intentaría mejorar las cotas de tiempo del sistema actual, aunque existen ciertos aspectos que son dependientes de la plataforma JADE, como es el caso de las búsquedas en el DF, las cuales deben asumirse.

- Una vez realizadas estas modificaciones, las cuales se orientan únicamente a mejorar el funcionamiento del sistema, pasaríamos a introducir nuevas funcionalidades, como por ejemplo, evaluar las cargas de cada tarea, de forma que a la hora de determinar el destino de una tarea, se tenga en cuenta el índice de carga de dicha tarea, o buscar algún método que ampliase la capacidad de ejecución de las tareas, de forma que no se deban ceñir únicamente a código java programado para ese fin.

5. REFERENCIAS

[1] Gómez Sanz, J. J., “Metodologías para el desarrollo de sistemas multi-agente”, Departamento de Sistemas Informáticos y Programación, Facultad de Informática, Universidad Complutense.

[2] Gomoluch y Schoroeder, “Information agents on the move: A survey on load-balancing with mobile agents”. Departamento de Computación, City University Londres. 2001, 12 Abril.

[3] “http://www.arpa.mil”

[4] “http://www.fipa.org”

[5] “http://www.jade.org”

[6] “http://www.mosix.org”

[7] “http://www.openmosix.org”

[8] Musen, M. A., “Modern Arquitectures for Intelligent Systems: Reusable Ontologies and Problem-Solving Methods” In C.G. Chute, Ed., AMIA Annual Symposium, Orlando, FL, 1998, pp. 46-52.

[9] Santo Orcero, D., “El Proyecto de Cluster SSI OpenMosix”, http://www.redes-linux.com/manuales/cluster/006.pdf

[11] Santo Orcero, D., “Los Clusters SSI”, Todo Linux, no. 23, pp. 57-60, 2002.

[12] Stefik, M., “Knowledge Systems”, Morgan Kaufmann Publishers, Inc., 1995.