Conciencia Tecnológica
ISSN: 1405-5597
contec@mail.ita.mx
Instituto Tecnológico de Aguascalientes
México

Luna Rosas, Fco. Javier; Tristán Ávila, Rene; Martínez Pedroza, J. de Jesús
Aplicando un Algoritmo Genético para Balancear Carga Dinámicamente en Ambientes Distribuidos Orientados a Objetos (CORBA)
Conciencia Tecnológica, núm. 23, 2003
Instituto Tecnológico de Aguascalientes
Aguascalientes, México

Aplicando un Algoritmo Genético para Balancear Carga Dinámicamente en Ambientes Distribuidos Orientados a Objetos (CORBA).

Fco. Javier Luna Rosas fjluna@verona.fi-p.unam.mx
Rene Tristán Ávila rtristana2@yahoo.mx
J. de Jesús Martínez Pedroza jmpedroza33@hotmail.com

División de Estudios de Posgrado Facultad de Ingenieria (DEPFI-UNAM), Cd. Universitaria, Apdo. Postal 70-256, C.P. 04510, México, D.F. Tels. (525) 5622-30-12 y (49) 9723515, Fax(525) 5616-10-73. Instituto Tecnológico de Aguascalientes, Av. A. López Mateos 1801 Ote. esq. Av. Tecnológico, Fracc. Ojocaliente, CP. 20256, Aguascalientes, Ags.

Resumen

Balancear Carga significa como distribuir procesos entre procesadores conectados por una red, para equilibrar la carga de trabajo entre ellos. Los algoritmos de planeación distribuida global pueden ser divididos en dos grandes grupos: algoritmos de balanceo de *carga dinámica y* algoritmos de balanceo de *carga estática.* Los algoritmos de balanceo de *carga estática*, también referenciados como planeación de tareas obtienen la localización de todos sus requerimientos antes de comenzar su ejecución. Los algoritmos de balanceo de *carga dinámica* intentan equilibrar la carga en tiempo de ejecución. Este artículo propone una nueva forma de balancear la carga dinámicamente aplicando *algoritmos genéticos* en ambientes distribuidos orientados a objetos CORBA sobre arquitecturas de demanda adaptativa, el artículo se centra en la manera de balancear carga dinámicamente en réplicas *homogéneas* utilizando el mismo tipo de procesador, requerimiento y velocidad de procesamiento y en réplicas *heterogéneas* aplicando diferentes procesadores y velocidades de procesamiento.

1. Taxonomía del Balanceo de Carga.

Los mecanismos de balanceo de carga distribuyen equitativamente la carga de trabajo de los clientes entre un conjunto de servidores para mejorar el tiempo de respuesta global del sistema. Los mecanismos de balanceo de carga generalmente se encuentran en uno de los niveles siguientes[9].

- A Nivel de Red.
- A Nivel del Sistema Operativo.
- A Nivel Middleware.

Balanceo de Carga Basado en Red: los servidores de nombres de dominio (DNS) y los ruteadores IP que sirven a una gran cantidad de máquinas host proveen este tipo de balanceo de carga. Por ejemplo cuando un cliente resuelve un hostname, el DNS puede asignar una dirección IP a cada requerimiento basado en las condiciones de carga actual, el cliente entonces contacta el servidor designado, sin que el cliente se entere, un servidor diferente puede ser designado en la próxima resolución de nombres. El balanceo de carga en esta capa es algo limitado, ya que no toman en cuenta el contenido de los requerimientos del cliente.

Balanceo de Carga Basado en Sistemas Operativos. los sistemas operativos distribuidos, proveen este tipo de balanceo de carga a través de un conjunto de computadoras, compartición de carga y mecanismos de migración de procesos. El agrupamiento es una forma efectiva de lograr alta disponibilidad y alto desempeño combinando muchas computadoras para mejorar el tiempo de respuesta global. Los procesos pueden entonces ser distribuidos transparentemente entre las computadoras de una agrupación. El agrupamiento generalmente emplea compartición de carga y migración de procesos.

Balanceo de Carga Basado en Middleware. Las interacciones globales son acopladas por las arquitecturas llamadas Middleware. La arquitectura más común de Middleware para aplicaciones orientadas a objetos distribuidas es Common Object Request Broker Architecture (CORBA) [7]. El balanceo de carga llevado a cabo en Middleware, frecuentemente es *Por-sesión* o *Por-requerimiento*. El balanceo de carga a nivel Middleware soportado por Object Request Brokers (ORBs) tal como CORBA, permite que los clientes invoquen operaciones sobre objetos distribuidos sin considerar localización de objetos, lenguajes de programación, plataforma de sistema operativo, protocolo de comunicación y hardware. Por otra parte, ORBs pueden determinar el requerimiento del cliente para dirigirlo a una replica específica.

Para ilustrar los beneficios del balanceo de carga a nivel Middleware, consideramos un sistema de inventarios en línea que se muestra en la Figura 1. Un sistema distribuido de inventarios en línea crea sesiones por las cuales se da el comercio. Este sistema consiste de múltiples servidores llamados réplicas los cuales procesan requerimientos enviados por los clientes en la red. Una réplica es un objeto que puede llevar a cabo la misma tarea que el objeto original. Las replicas de servidores que hacen las mismas operaciones pueden ser agrupadas dentro de un grupo de servidores, el cual es también conocido como grupo de replicas o grupo de objetos. Para el ejemplo de la Figura 1, una réplica es una manera de reducir la carga de cualquier servidor. La carga en este caso es la combinación del promedio del número de sesiones con requerimiento por unidad de tiempo, y el monto total de recursos empleados actualmente para atender la sesión. Las cargas son balanceadas a través de réplicas que están agrupadas. Las réplicas no necesitan estar en la misma localización física. El propósito de la réplica es crear una sesión de negociación del inventario. En este tipo de sistema los requerimientos del cliente llegan de manera dinámica (no determinística) y la duración de cada requerimiento no tiene un conocimiento a priori.

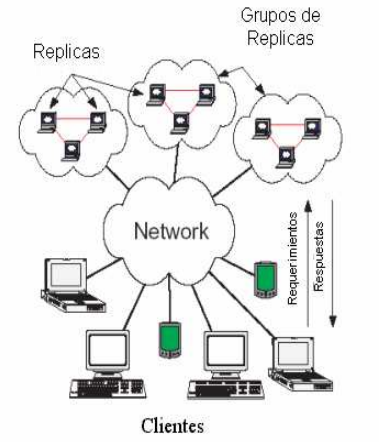

Figura 1. Sistema de Inventarios en Línea.

2. Arquitecturas y Estrategias de Balanceo de Carga en CORBA.

Hay varias estrategias para diseñar en CORBA servicios de balanceo de carga [9]. Estas estrategias pueden ser clasificadas de la siguiente manera:

Por-sesión: Los requerimientos del cliente continuarán siendo enviados a la misma réplica durante la sesión (en el contexto de CORBA, una sesión se define como el período de tiempo que un cliente es conectado al servidor con el propósito de invocar operaciones remotas de objetos en el servidor). La arquitectura se define por el periodo de vida de los clientes.

Por-requerimiento: Cada requerimiento del cliente puede ser enviado a una replica diferente, esto es, se realizara el atado del requerimiento a una réplica cada vez que es invocado. La arquitectura se define por el periodo de vida de los requerimientos.
Sobre-demanda: Los requerimientos del cliente son enviados en conjuntos y atados a una réplica seleccionada de acuerdo al algoritmo de balanceo de carga que toma como base el estado global del sistema. La arquitectura de balanceo se define por el atado de un conjunto de requerimientos.

Política de balanceo: Cuando se diseña un servicio de balanceo de carga es importante seleccionar un algoritmo adecuado que decida que replica procesará el requerimiento que llega. Por ejemplo aplicaciones donde todos los requerimientos generados de carga son muy semejantes pueden usar un simple algoritmo round-robin, mientras aplicaciones donde la carga generada por cada uno de los requerimientos no puede ser predecida, requiera de algoritmos más complejos. En general, las políticas de balanceo de carga pueden ser clasificadas dentro de las categorías siguientes:

No adaptativa: Un balanceador de carga puede usar una políticas *No adaptativa* para el atado de requerimientos y esa política es aplicada por toda la vida del cliente. El algoritmo seleccionado puede ser tan simple como el round-robin o el aleatorio (random) para seleccionar una replica en donde se procesará el requerimiento.

Adaptativa: Un balanceador de carga puede usar políticas *Adaptativas* que utilicen información en tiempo de ejecución (réplicas disponibles, carga de trabajo en las réplicas, requerimientos en espera, etc.) para seleccionar la mejor réplica, que

procese sus requerimientos, o bien, cambiar a otra replica cuando esta no proporcione el resultado esperado por el sistema.

Combinando las estrategias descritas anteriormente de varias maneras, es posible crear varias arquitecturas de balanceo de carga.

No adaptativa Por-sesión

Una manera de diseñar una Arquitectura de balanceo de carga en CORBA es hacer que el balanceador de carga seleccione la réplica destino donde una sesión cliente/servidor quedara establecida, es decir, cuando un cliente obtiene una referencia de objeto a un objeto CORBA (nombre de la réplica) y se conecta a ese objeto. Tal como se muestra en la Figura 2.

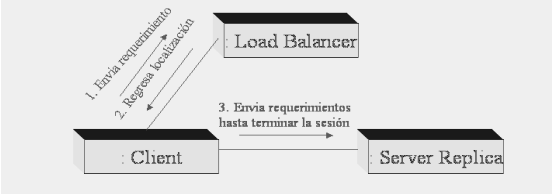

Figura 2 .Arquitectura No adaptativa Por-sesión

Note que la política de balanceo en esta arquitectura es *No adaptativa* ya que el cliente interactúa con el mismo servidor al cual este fue atado originalmente. Esta arquitectura es adecuada para la política de balanceo de carga que implementa algoritmo round-robin o random.

No adaptativa Por-requerimiento

Una arquitectura *No adaptativa* Por-requerimiento comparte muchas características de la arquitectura *No adaptativa* Por-sesión. La primera diferencia es que el cliente es atado a la réplica cada vez que un requerimiento es invocado. Esta arquitectura tiene la desventaja de degradar el desempeño del sistema debido al incremento de sobre carga (overhead) en la comunicación.

No adaptativa Sobre-demanda

Tiene las mismas características Por-sesión. Sin embargo, la arquitectura *No adaptativa* sobre-demanda permite reorganizar los requerimientos del cliente de una manera arbitraria en cualquier tiempo. La información en tiempo de ejecución, como la carga de cada réplica, no es usada para decidir cuando se reorganizan los clientes. En lugar de eso, los clientes se pueden reorganizar en intervalos regulares de tiempo.

Adaptativa Por-sesión

Esta arquitectura es similar a *la No adaptativa* Por-sesión. La primera diferencia es que en una arquitectura *Adaptativa* Por-sesión puede usar información de la carga en tiempo de ejecución para seleccionar la réplica, por esa razón, es menos probable atar un cliente a un servidor con sobre carga. Sin embargo, la carga generada por los clientes pueden cambiar después de hacer la decisión de atado.

Adaptativa Por-requerimiento

La Figura 3 muestra una Arquitectura de Balanceo de Carga *Adaptativa* Por-requerimiento, este diseño introduce un front-end, el cual es un servidor proxy que recibe todos los requerimientos de los clientes. En este caso el servidor “front-end” es el balanceador de carga. El balanceador de carga selecciona una réplica apropiada de acuerdo con las políticas de balanceo de carga y envía el requerimiento a la replica. El front-end espera la respuesta de las replicas para enviarla a los clientes.

Adaptativa sobre-demanda

Esta arquitectura se muestra en la Figura 4. Los clientes reciben un objeto de referencia al balanceador de carga inicial. Usando un mecanismo estándar de CORBA, el balanceador de carga puede redirigir el requerimiento del cliente inicial a la

apropiada réplica. Los clientes de CORBA continuarán usando el nuevo objeto de referencia y se comunicaran con las replicas directamente hasta que ellos sean redireccionados otra vez o terminen su sesión.

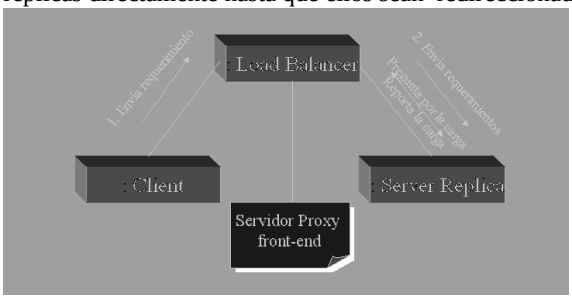

Figura 3. Arquitectura Adaptativa Por-sesión

Después de definir las políticas y estrategias de balanceo de carga, con sus ventajas e inconvenientes, deberán estar soportadas en una arquitectura que idealmente deben de responder a las siguientes características[8]:

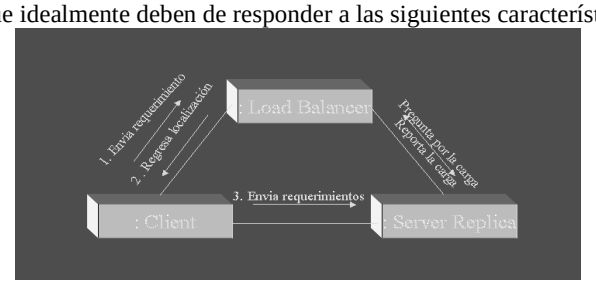

Figura 4. Arquitectura Adaptativa sobre-demanda

- Transparencia de servidores
- Balanceo de carga descentralizado.
- Conservación del estado de las réplicas
- Monitoreo de carga distribuido.
- Activación de replicas sobre-demanda
- Balanceo de carga tolerante a fallas
- Soporte a distintos algoritmos de balanceo de carga.

Soporte a distintos algoritmos de balanceo de carga. Muchas arquitecturas de balancear carga solamente soportan un tipo de algoritmo para balancear carga. Esos algoritmos pudieran no ser los adecuados todo el tiempo durante el ciclo de vida de una aplicación distribuida. Peor aún esos algoritmos pudieran estar configurados estáticamente en un servicio de balanceo de carga. Si el trafico del cliente cambia sustancialmente en tiempo de ejecución, la carga de las réplicas, no será balanceada efectivamente. Hay ocasiones en que las arquitecturas de balanceo de carga (tales como TAO) utilizan patrones para adaptar el algoritmo de balanceo de carga en tiempo de ejecución [10]. Pero generalmente los algoritmos que usan una buen estrategia de balanceo de carga siempre terminan haciendo un intercambio (tradeoff) entre desempeño y costo, ya que mantienen demasiada comunicación entre cada uno de sus componentes provocando carga excesiva en la red y en el procesador (overhead). Si mantenemos conocimiento a priori en la estrategia de balanceo de carga, podemos tener un mejor rendimiento entre desempeño y costo. Un algoritmo genético puede darnos un buen desempeño en este tipo de problemas ya que mantiene conocimiento casi optimo de la estrategia de balanceo de carga.

3. Balanceando la Carga con Algoritmos Genéticos.

La naturaleza utiliza potentes medios para impulsar la evolución satisfactoria de los organismos. Los organismos que son poco aptos para un determinado ambiente mueren, en tanto los que están bien adaptados para vivir se reproducen.

method GENETIC-ALGORITHM (*population, fitness*)

return
an individual.
inputs:
population, a set from many individual.
fitness, is the quantity that determines the quality of a chromosome.
repit
parents ← selection(*population, fitness*)
new_*population* ← createChilds (*parents*)
until
one individual is good.

Figura 5. Algoritmo Genético con Evolución Simulada.

Los hijos son semejantes a sus padres, por lo que cada nueva generación tiene organismos semejantes a los miembros bien dotados de la generación anterior. Ocasionalmente se producen mutaciones al azar, y aunque implica la pronta muerte del individuo mutado, algunas de estas mutaciones dan como resultado nuevas y satisfactorias especies. En la figura 5 mostramos el algoritmo genético, que empieza por un conjunto de uno o varios individuos y aplica operadores de selección y reproducción para que evolucione un individuo satisfactoriamente medido por una función de adaptación.

3.1 Longitud de los Genes.

Bigus [1] dice que cada población tiene una gran cantidad de genes, consistiendo de un gran numero de cromosomas generados por el proceso de selección natural, los cromosomas generalmente tienen una representación binaria, pero pudieran ser valores reales, estos cromosomas deben ser evaluados en forma aleatoria mediante algún proceso de cruzamiento para generar nuevos cromosomas, los cromosomas con mejores características sobreviven, en tanto que los cromosomas con pobres características son descartados. Nosotros generamos nuestra población inicial de manera aleatoria, cada cromosoma de

$$BPR = Log(Length(\text{Re } plicas)) / Log(2).$$

la población esta compuesto de varios genes que son construidos en base al número de réplicas de la siguiente manera:

BPR = Numero de bits por réplica.
Log(Length(Replicas)) = Logaritmo natural en base al número de réplicas.
Log(2) = Logaritmo base 2.

Cabe hacer mención que el escalamiento de las réplicas es base 2. La Tabla 1 muestra cada uno de los genes que representa la réplica donde uno o más requerimientos serán procesados. Cada gene es representado en 2 a la B Bits + 1.

Numero Decimal	Representación Binaria (Genes)	Réplica 2^B+1
0	000	1
1	001	2
2	010	3
3	011	4
4	100	5
5	101	6
6	110	7
7	111	8

Tabla 1. Representación de las Réplicas en Base 2 + 1.

En general, B bits son suficientes para representar 2 a la B diferentes enteros, esto es el número de bits B requeridos para representar N consecutivas réplicas satisface la ecuación siguiente:

$$2^B + 1 \geq N$$

3.2 Composición del Cromosoma.

Los strings o individuos (cromosomas), son arreglos de bits que son seleccionados de manera aleatoria para sobrevivir y procrear la próxima generación de cromosomas. Nuestros cromosomas están compuestos en función del numero de requerimientos y el número de réplicas. Una vez calculado el tamaño del gene el siguiente paso es calcular la longitud del cromosoma que esta dado por la formula siguiente:

$$L\_Cromosoma = BPR * N\_\text{Re}querimientos$$

L_Cromosoma. Longitud del Cromosoma.
N_Requerimientos. Número de Requerimientos.

Así, por ejemplo, si suponemos que tenemos 25 requerimientos y 8 réplicas posibles para procesar esos requerimientos, la longitud del cromosoma (L_Cromosoma) estará compuesto por el producto, que es el tamaño del gene (compuesto de 3 bits, dado que tenemos 8 réplicas posibles) multiplicado por el número de requerimientos, por lo tanto la longitud del cromosoma para este caso especifico será de 75 bits.

3.3 Generando la Población Inicial.

Un algoritmo genético comienza con una gran comunidad de cromosomas conocida como la población inicial. La población inicial tiene un número de cromosomas inicial (Nipop), cada cromosoma esta constituido de ceros y unos que son aleatoriamente seleccionados, en general la población inicial la podemos calcular con la formula siguiente:

random(Nipop,Nbits) = Es una matriz de Nipop X Nbits.

Cada renglón en la matriz es un cromosoma, el cromosoma corresponde a un valor discreto de cierta longitud, luego cada cromosoma es pasado como un parámetro para evaluar la función objetivo (fitness).

4. Aplicando el Ciclo Genético.

4.1 El Proceso Evolutivo.

La Figura 6 muestra la manera en que aplicamos el proceso evolutivo utilizando dos cromosomas padres, A y B con ejemplos de mutación y un operador de cruce en un punto (one-point crossover). Primero se realiza una mutación en el cromosoma padre A cambiando la cuarta posición de 0 a 1 y la novena posición de 1 a 0 partiendo de izquierda a derecha del cromosoma, los bits cambiados son indicados por un subrayado, enseguida aplicamos un operador crossover a los padres en la tercera posición: tomamos los tres bits del padre A y los 7 bits derechos del padre B para crear el hijo 1, y los 3 bits del padre B con los 7 bits del lado derecho del padre A y creamos el hijo 2. Este proceso evolutivo es aplicado en nuestro balanceo de carga con el fin de obtener los mejores cromosomas que me indiquen un estado ideal de balanceo, basándonos por supuesto en la función objetivo que será descrita posteriormente.

Parent A:		0010010011
Parent B:		1100001010
Mutate(A)		0011010001
Crossover(A,B)	Child 1 :	0010001010
	Child 2 :	1100010011

Figura 6. Cruzamiento (Crossover) y Mutación (Mutate).

4.2 Función Objetivo (fitness).

$$Nipop = random(Nipop, Nbits)$$

La función objetivo de un algoritmo genético puede ser diseñada para ejecutar diferentes tareas de búsqueda de optimización. El valor de la función objetivo proporciona la cantidad para guiar el proceso de reproducción de los algoritmos genéticos a una nueva generación. Usualmente la función objetivo es diseñada de la forma que los valores sean todos positivos y el cromosoma que tenga el valor mas alto es el que genera el mejor desempeño en toda la población. Construimos la función objetivo tratando de minimizar la máxima diferencia entre longitudes de requerimientos dentro de las réplicas, es decir el

valor máximo que existe entre la réplica que tiene el mayor número de requerimientos y la réplica que tiene el menor número de requerimientos, este valor es sumado a la mayor longitud de requerimientos, una vez teniendo la suma se saca el reciproco de la función tratando de minimizar el valor máximo de la función de la forma siguiente:

1 / (abs(max(Lqs_b)) + abs(max (diff(Lqs_b))))

abs(max(diff(Lqs_b))). Máxima diferencia que hay entre longitudes de requerimientos de las réplicas.

abs(max(Lqs_b)). La réplica que mantiene el mayor número de requerimientos.

La Figura 7 muestra la convergencia de las poblaciones, cuando alcanzamos la máxima convergencia entre las poblaciones obtenemos los mejores cromosomas que nos permiten indicar la forma en que podemos distribuir los requerimientos de carga de una manera optima. La línea inferior muestra la población anterior, la línea superior muestra la población mejorada después de que la función objetivo ha sido evaluada y los cromosomas con los valores mas altos han sido tomados para guiar el proceso de reproducción a la nueva generación. Cabe hacer mención que se han seleccionado solo los mejores individuos de la población, es decir, se selección la mitad de individuos de la población para formar la próxima generación genética. La grafica de la figura 7 muestra la forma en que van convergiendo los mejores individuos de la población anterior con los mejores individuos de la población actual, en ella vemos que se llega a un punto de convergencia en un determinado número de generaciones, cuando ya no existe diferencia alguna entre las generaciones la grafica se mantiene uniforme, obteniendo así los mejores cromosomas, que nos permitirán una distribución optima de requerimientos.

5. Estrategias y Resultados.

Existen ocasiones en que las técnicas de carga empleadas no siempre son las adecuadas en todas las situaciones de carga, contar con estrategias que permitan adecuar la carga de los requerimientos de manera dinámica es el objetivo principal de este trabajo, las estrategias se realizaron de la siguiente manera:

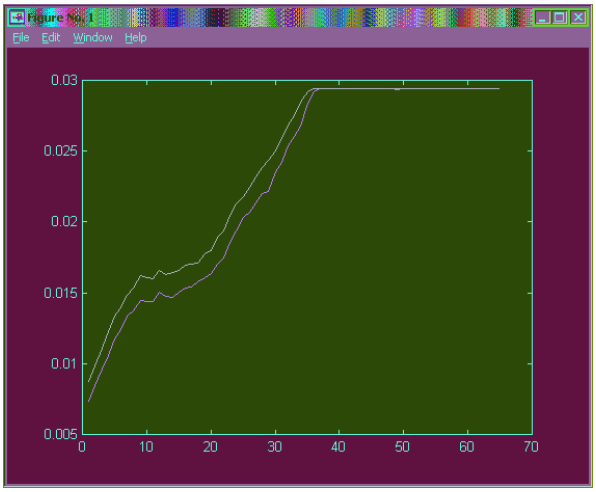

Figura 7. Convergencia de la Función Objetivo (Fitness), Aplicando Balanceo de Carga Homogéneo.

Primero. Cada cromosoma de la población se construyó tomando como base el número de replicas disponibles.
Segundo. De toda la población se eligieron los M/2 individuos que tienen una mejor evaluación.
Tercero. Se tomo en cuenta el balanceo de carga homogéneo, es decir se utilizo un mismo tipo de requerimiento, arquitectura y velocidad de procesamiento. Las figuras 8 y 9 muestran la manera en que balanceamos homogéneamente 40 requerimientos en 8 réplicas utilizando una población de 100 cromosomas y 65 generaciones.

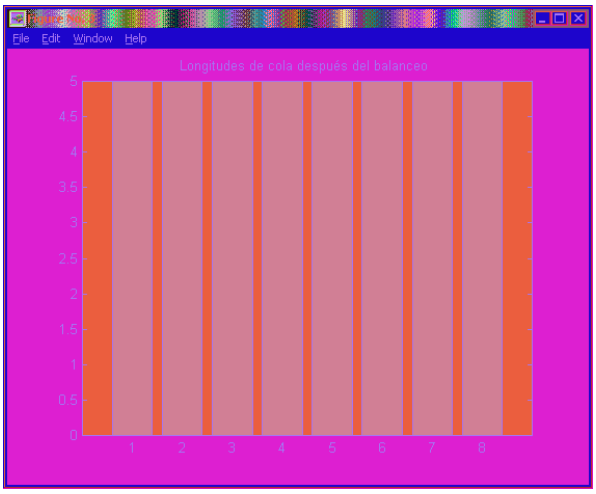

Figura 8. Balanceo de Carga Homogéneo, Utilizando una Carga de 40 Requerimientos, 8 Replicas, 100 cromosomas y 65 generaciones.

Cuarto. El balanceo de carga heterogéneo se realizo tomando en consideración diferentes velocidades de procesamiento y diferentes tipos de requerimientos. La figura 10 nos muestran la manera de balancear carga heterogéneamente utilizando 40 requerimientos, 8 réplicas, 100 cromosomas y 65 generaciones. Los servidores 6 y 8 son los que tienen mayor velocidad por lo tanto el algoritmo genético intenta proporcionarles mayor carga, los servidores 2, 4, 5 y 7 mantienen la misma velocidad, pero su procesamiento es mas lento que los servidores restantes por lo que el algoritmo genético siempre intenta mantener las colas de procesamiento lo más pequeñas posibles con un mismo equilibrio de respuesta. Por ultimo vemos los servidores 1 y 3 que mantienen una velocidad intermedia, pero siempre intentando mantener las colas de procesamiento equilibradas con un mismo tiempo de respuesta.

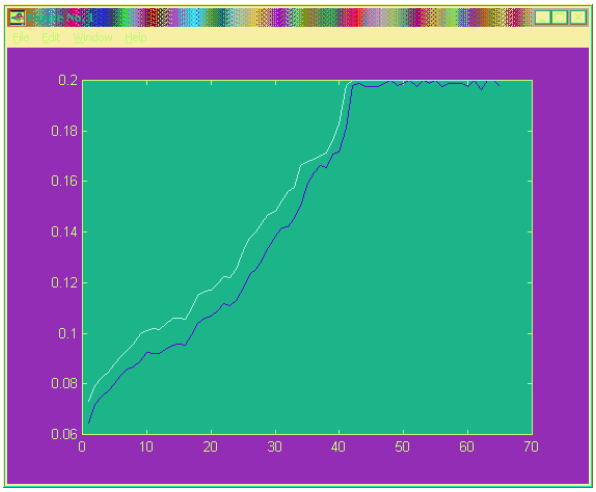

Figura 9. Convergencia de la Función Objetivo (Fitness). Aplicando Balanceo de Carga Heterogénea.

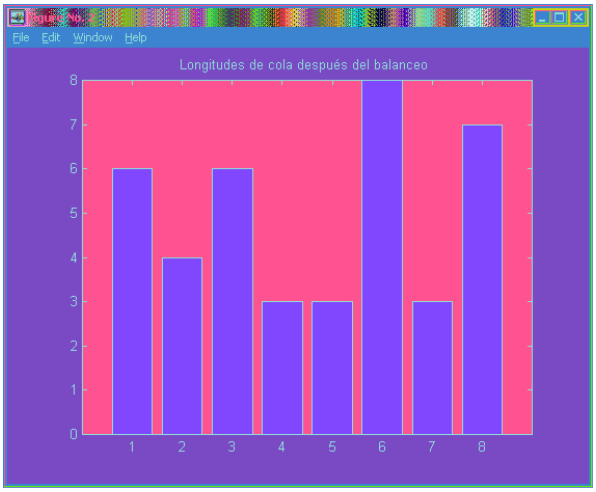

Figura 10. Balanceo de Carga Heterogéneo, Utilizando una Carga de 40 Requerimientos, 8 Réplicas con Arquitecturas de Procesamiento Heterogéneas, 100 Cromosomas y 65 Generaciones.

6. Conclusiones.

Un sistema distribuido emplea el servicio de balanceo de carga para mejorar el rendimiento asegurándose que las cargas entre las réplicas sean lo mas uniforme posible. En algunas aplicaciones hay veces en que podemos predecir los picos de carga que suceden en un día, o en varios días en una semana. En otras aplicaciones la carga no puede predecirse tan fácilmente (no se tiene conocimiento a priori de las cargas). Una buena estrategia para balancear carga dinámicamente es usar un algoritmo de balanceo de carga que permita ecualizar la carga entre diferentes réplicas, pero hay ocasiones en que las condiciones de carga cambian drásticamente y el algoritmo de balanceo de carga ya no es suficiente para adaptarse a las condiciones de carga actual, contar con un algoritmo que mantenga conocimiento a priori de todos los casos posibles de carga es el objetivo principal de este trabajo, en lo descrito anteriormente solo describimos un caso especifico de carga y la forma de balancearla aplicando un algoritmo genético, seguramente conforme avancemos en el estado del arte podremos describir todos los casos posibles de balanceo de carga.

Bibliografía.

[1] Bigus P. Joseph and jennifer, Constructing Intelligent Agents with Java. A Programmer´s Guide to Smarter Applications. John Wiley & Sons, Inc. 2001.

[2] Eager, D. “Adaptive Load Sharing in Homogeneous Distributed Systems”. IEEE Transactions on Software Engineering, may 1986.

[3] Goldberg David E. Genetic Algorithms, in Search, Optimization and Machine Learning. Addison Wesley 1989.

[4] Goscinski, Andrzej., Distributed Operating Systems. The Logical Design. Addison-Wesley 1992.

[5] Haupt L. Randy, Haupt Sue Ellen. Practical Genetic Algorithms. Wiley 1998.

[6] Hisao Kameda, Jie Li, Chonggun Kim and Yongbing Zhang, Optimal Load Balancing in Distributed Computer System. Springer-Verlag, London 1997.

[7] OMG (Object Management Group). The Common Object Request Broker: Architecture and Specification. Technical Report, http:// www.omg.org(2001).

[8] Ossama Othman, O´Ryan Carlos and Schmidt. An Efficient Adaptive Load Balancing Service for CORBA. ACM 2001.

[9] Ossama Othman, O´Ryan Carlos and Schmidt. Strategies for CORBA Middleware-Based Load Balancing. IEEE Distributed Systems Online http://computer.org/dsonline/0103/features/oth103_1.html

[10] Ossama Othman, O´Ryan Carlos and Schmidt C. Designing an Adaptive CORBA Load Balancing Service Using TAO. IEEE Distributed Systems Online http://computer.org/dsonline/0104/features/oth104_print.html

[11] Russell Stuart, Norving Peter. Inteligencia Artificial un Enfoque Moderno. Prentice may 1996.

[12] Singhal, Mukesh and Shivaratri G. Niranjan, "ADVANCED CONCEPTS IN OPERATING SYSTEMS Distributed, Database, and Multiprocessor Operating Systems", McGraw-Hill, 1994.

[13] Tsoukalas Lefteri, Uhrig E. Robert, Fuzzy and Neural Approaches in Engineering, Wiley 1997].