Conciencia Tecnológica No. 44, Julio-Diciembre 2012

Combinando Least-Loaded Más una Estrategia Heurística para Optimizar el Tiempo de Respuesta Total Bajo CORBA

Investigación

M. C. Gricelda Medina Veloz[1] Dr. Francisco Javier Luna Rosas[2] Dr. Julio César Martínez Romo[2]
M. C. J. Valentín López Rivas[2] Dr. Carlos Alejandro de Luna Ortega[3]
[1]Universidad Tecnológica del Norte de Aguascalientes, Av. Universidad 1001, Estación Rincón, C.P. 20420, Rincón de Romos, Ags.e-mail:grismv2000@yahoo.com
[2]Instituto Tecnológico de Aguascalientes, Av. A. López Mateos 1801 Ote. esq. Av. Tecnológico, Fracc. Ojocaliente, CP. 20256, Aguascalientes, Ags.
[3]Universidad Politécnica de Aguascalientes, Departamento de Ingeniería Mecatrónica. Prolongación Mahatma Gandhi Km 2 Aguascalientes Ags. CP 2100.

Resumen

En este artículo proponemos una nueva forma de optimizar dinámicamente la carga combinando en forma adaptativa una estrategia heurística en conjunto con Least_Loaded (LL), el trabajo se centra en la manera de combinar la estrategia heurística y LL con el propósito de mejorar el tiempo de respuesta total de una aplicación. Implementamos las estrategias heurísticas y LL en una arquitectura para optimizar carga bajo el estándar de CORBA (Common Object Request Broker Architecture). Realizamos evaluaciones utilizando diferentes métricas de carga e ilustramos como la estrategia LL en conjunto con la estrategia evolutiva bajo ciertas condiciones del sistema arrojó mejores resultados.

Palabras clave: optimización de carga, heurística, CORBA, tiempo de respuesta total, patrón estrategia.

Abstract

In this article we propose a new way to optimize the load dynamically, combining a heuristic strategy with Least_Loaded (LL). Our work is centered in the way of to combine heuristic strategy and LL with the purpose of improving the total time of an application. We implemented the heuristics strategies and LL in architecture for load optimizing under the CORBA standard. We evaluated using different load metrics, and we show that under certain conditions of the system the LL strategy with the evolutionary strategy obtained better results.

Key words: load optimization, heuristics, total response time, strategy pattern.

Introducción

Una arquitectura de optimización de carga es un sistema que permite distribuir el trabajo computacional entre varias máquinas, con el objetivo de reducir el tiempo de respuesta global del sistema.

Las arquitecturas de optimización de carga forman parte de los sistemas distribuidos, los sistemas distribuidos están compuestos por servidores que son programas que están en ejecución en una computadora de la red, aceptando peticiones de otro programa que se ejecuta en otras computadoras, denominadas clientes, cuya finalidad es esperar que se procese dicha petición y recibir una respuesta adecuada. Los servidores están en ejecución continuamente, y los clientes lo hacen solamente cuando están realizando invocaciones remotas [1].

Una arquitectura de optimización siempre utiliza una estrategia de optimización o balanceo de carga, las estrategias de optimización de carga pueden ser divididas en dos grandes grupos: estrategias de optimización estáticas y estrategias de optimización dinámicas. Las estrategias de optimización estáticas, obtienen la localización de todos sus procesos antes de comenzar la ejecución. Las estrategias de optimización dinámicas intentan equilibrar la carga en tiempo de ejecución [1].

Cuando se hace optimización de carga se aplica una técnica bien establecida para utilizar los recursos de computación disponibles más efectivamente, las aplicaciones distribuidas pueden mejorar su escalabilidad, tiempo de respuesta y uso de recursos empleando optimización de carga en varias formas y en varios niveles del sistema, incluyendo la red, el sistema operativo y a nivel middleware [2].

La arquitectura más común de middleware para aplicaciones orientadas a objetos distribuidas es Common Object Request Broker Architecture (CORBA) [3].

Optimización de Carga Bajo CORBA

Hay varias estrategias y políticas para diseñar en CORBA servicios de optimización de carga [4], [5], [6]. 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 réplica diferente, esto es, se realizará un atado del requerimiento a la réplica por cada requerimiento del cliente. 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 optimización de carga que toma como base el estado global del sistema. La arquitectura de optimización se define por el atado de un conjunto de requerimientos.

Por otra parte las políticas de optimización de carga pueden ser clasificadas dentro de las categorías siguientes:

No adaptativas: Un optimizador de carga puede usar una política n*o 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-Robín o el Aleatorio (Random) para seleccionar una réplica en donde se procesará el requerimiento.

Adaptativas: Un optimizador de carga puede usar políticas a*daptativas* 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 réplica cuando esta no proporcione el resultado esperado por el sistema.

Si combinamos las estrategias con las políticas anteriormente descritas de varias maneras, es posible crear las siguientes arquitecturas de optimización de carga: 1) No adaptativa Por-sesión, 2) No adaptativa Por-requerimiento, 3) No adaptativa Sobre-demanda, 4) Adaptativa Por-sesión, 5) Adaptativa Por-requerimiento, 6) Adaptativa Por-demanda.

Para consolidar el enfoque previo y resolver este problema la OMG u Object Management Group, diseñó el Request For Proposal (Requerimiento de extensión de funcionalidades de CORBA) para establecer un estándar de balanceo de carga y monitoreo de sus aplicaciones. El artículo de IONA Technologies [7] fue considerado el trabajo más relevante en esta área y la OMG recomendó esta adopción. Este artículo incluye tres estrategias que deberán soportar las implementaciones: dos estrategias estáticas (Round-Robin y Random) más una estrategia dinámica (Least-Loaded (LL)). La estrategia dinámica potencialmente se desempeña mejor que la estática porque ella considera la información de estado actual para hacer decisiones de optimización de carga.

Como mencionamos previamente este trabajo propone una nueva forma de optimizar dinámicamente la carga bajo CORBA combinando una estrategia Heurística en conjunto con Least_Loaded (LL), como LL ya fue evaluada y comparada en [6], el trabajo se centra en la manera de combinar la estrategia Heurística y LL con el propósito de mejorar la optimización dinámica de LL en CORBA. Con la intención de detectar las mejores condiciones del sistema, se evaluaron las estrategias realizando optimización dinámica en diversas aplicaciones de transferencias de archivos. Una vez detectadas las mejores condiciones del sistema pretendemos evaluar en un futuro aplicaciones que involucren gran tiempo de procesamiento, por ejemplo, en el reconocimiento y verificación de firmas manuscritas On-Line y Off-Line (en línea y fuera de línea) en donde el tiempo de respuesta es sumamente demandante.

El resto de este artículo está organizado como sigue: en la sección 2 describimos la arquitectura para optimizar carga bajo CORBA, y en la sección 3 cómo implementar las estrategias en conjunto con LL para optimizar carga dinámicamente en ambientes distribuidos orientados a objetos (CORBA), en la sección 4 mostramos el análisis comparativo y los resultados de las estrategias de optimización de carga para finalmente en la sección 5 dar nuestras conclusiones.

Diseño dela Arquitectura de Optimización de Carga Bajo CORBA

El Lenguaje Unificado de Modelado (UML, Unified Modeling Language) es un lenguaje gráfico para visualizar, especificar, construir y documentar modelos que se fundamenten en la tecnología orientada a objetos [8], [9], [22]. Utilizamos UML como herramienta de

modelado para realizar el análisis, diseño y construcción de la arquitectura, con el objetivo de construir modelos que cubran diferentes vistas y comportamientos que ayuden a generar los planos estándar que nos lleven a la automatización directa de la arquitectura.

Cabe hacer mención que en dos trabajos anteriores hemos realizado un análisis más detallado de la arquitectura de optimización de carga [10], [11] sin embargo vamos a retomar los componentes de la arquitectura, para visualizar mejor el trabajo propuesto. También resaltamos en este escrito que optimización de carga y balanceo de carga, son dos conceptos similares y serán usados indistintamente.

En general la arquitectura está compuesta de los siguientes componentes:

- ***LBA_Analizador.*** No es común para todas las aplicaciones distribuidas exhibir las mismas condiciones de carga, significa que algunas estrategias de optimización de carga son más aplicables a algunos tipos de aplicaciones que a otras. Un optimizador de carga debe ser lo suficientemente flexible para soportar diferentes tipos de estrategias de optimización de carga. Esas estrategias de optimización de carga son encapsuladas en el componente LBA_Analyzer. La propuesta del LBA_ Analyzer es tan evidente como su nombre lo indica, el objetivo de este componente es determinar las condiciones de carga sobre las réplicas o grupos de réplicas registradas en el optimizador de carga, dependiendo de la aplicación el LBA_Analyzer escoge la estrategia que mejor se adapte al grupo de réplicas registrado.
- ***LBA_Monitor.*** El componente LBA_ Monitor se encarga de monitorear la carga de una réplica dada y de revisar el buen funcionamiento de las réplicas en tiempo de ejecución. Además de reportar la carga de las réplicas al analizador de carga usando una política en modo push (Push Policy).
- ***LBA_Réplica***. Este componente se encarga de procesar los requerimientos de los clientes y representa también los servidores que proporcionan respuestas a los mismos. Una manera de hacer transparente el envío de los requerimientos de los clientes es utilizando LOCATION_FORWARD implementado en CORBA (ver Figura 2.1).

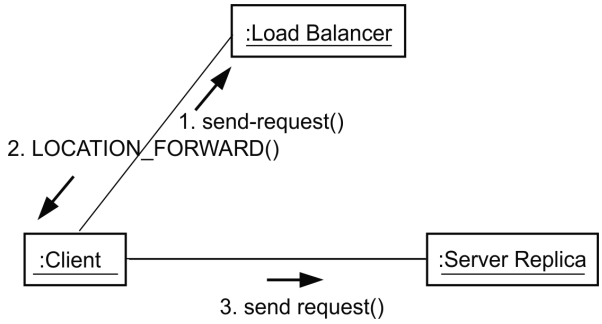

Figura 2.1. Direccionamiento Transparente en el Envío de Requerimientos de los Clientes [3].

- ***LBA_Cliente.*** Este componente es el encargado de solicitar los requerimientos y pedir que se les dé servicio a estos.
- ***Estrategia.*** Este componente se encarga de establecer la métrica de carga que utilizará la arquitectura (descrita posteriormente en este apartado).
- ***Balanceador.*** Este componente liga todos los componentes anteriormente descritos.

Los diagramas de interacción son modelos que describen la manera en que colaboran grupos de objetos para obtener cierto comportamiento [12]. La arquitectura de optimización de carga fue desarrollada basada en este tipo de diagramas. El diagrama de la Figura 2.2 muestra la secuencia de operaciones que ocurre cuando el optimizador de carga obliga a una réplica sobrecargada a rechazar los requerimientos de los clientes y a su vez elige una réplica que este en posibilidades de procesar estos requerimientos.

1. El cliente solicita al optimizador de carga (load balancer) una réplica para procesar sus requerimientos.
2. Después de recibir los requerimientos del cliente, el optimizador los envía al componente analizador (analyzer).
6. El monitor revisa el estado que guardan las réplicas y lo envía al analizador.
7. El analizador elige la estrategia para optimizar la carga.
8. El analizador le dice a la réplica que rechace los requerimientos que le sean enviados.
9. El analizador de carga vuelve a elegir una nueva réplica, aplicando nuevamente la política de optimizar carga.

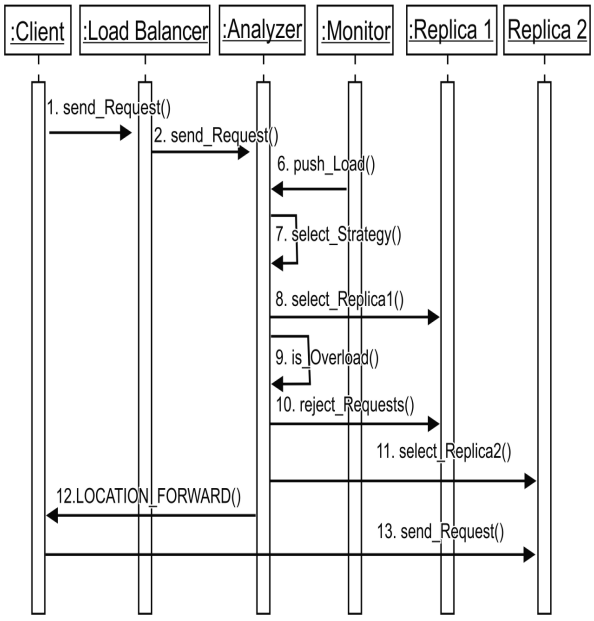

Figura 2.2. La Arquitectura de Optimización de Carga Elige una Réplica Donde Procesar Requerimientos.

10. Si la réplica se encuentra en posibilidades de procesar los requerimientos de los clientes, se retorna el mensaje LOCATION_FORWARD para redireccionar (rebind) los requerimientos de los clientes a la nueva réplica.
11. Una vez que el optimizador de carga encuentra la réplica, se indica al cliente que réplica deberá atender sus requerimientos y este solicita se les dé respuesta.

Diseño de la Estrategias en Conjunto con LL

El proceso de desarrollo de software moderno incluye el diseño de patrones de software los cuales son una descripción formal de buenas soluciones a problemas ya planteados anteriormente, estos pueden ser usados por los desarrolladores de software como una colección de conocimiento experto acerca de un problema específico. Un amplio rango de patrones de diseño puede ser obtenido, en [13], [14].

Diseño del Patrón Estrategia

En general, no es común que todas las aplicaciones distribuidas exhiban las mismas condiciones de carga, significa que algunas estrategias de balanceo son más aplicables a algunos tipos de aplicaciones que a otras. Un optimizador de carga debe ser lo suficientemente flexible para soportar diferentes tipos de estrategias de optimización Figura 3.1.

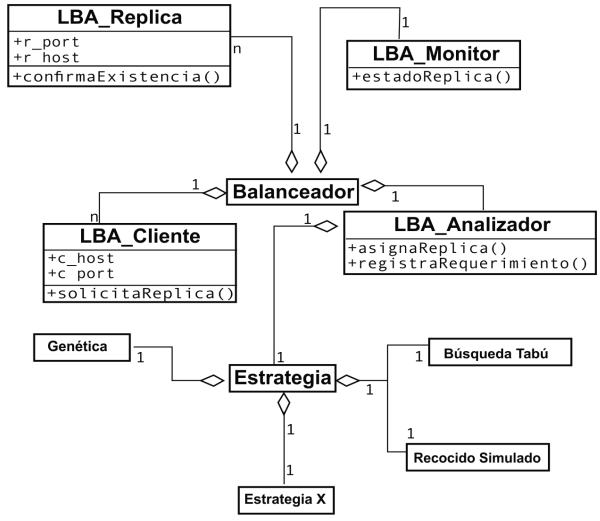

Figura 3.1 Estructura Estática del Patrón Estrategia.

Cabe hacer mención nuevamente que la idea del patrón estrategia así como algunos otros patrones de nuestra arquitectura fueron utilizados de nuestros trabajos ya mencionados previamente. El objetivo de los patrones de software es facilitar la reutilización del diseño para lograr crear un lenguaje común de comunicación entre los desarrolladores además de promover el uso de buenas prácticas en el proceso de diseño y construcción de software.

Combinando Least_Loaded Más una Estrategia Heurística para Optimizar el Tiempo de Respuesta Total

Como LL se encuentra ampliamente explicada en diversos artículos de la literatura [7], [15], este trabajo se centra en dar una explicación basada en el diseño de la estrategia de optimización de carga implementando diversas heurísticas. La mejor forma de resolver un problema de planeación de carga es centrar el problema como un problema de optimización [16]. En la última década, los Algoritmos Evolutivos (Genéticos), Recocido Simulado (SA), Búsqueda Tabú (TS) han sido extensamente usados como herramientas de búsqueda y optimización en varios dominios de problemas, incluyendo las ciencias, el comercio y la ingeniería. La razón primaria de su éxito es la amplia aplicabilidad, facilidad de uso y perspectiva global [17] [18] [19] [20] [21].
Un simple ejemplo es mostrado para ilustrar cómo nuestro mecanismo de optimización de carga trabaja. Todos los pasos comienzan desde el momento que el mecanismo de optimización de carga es inicializado hasta que los requerimientos son asignados.

Paso 1. Asignación de Carga Sin Estrategia. Antes de que la estrategia de optimización entre en acción el analizador selecciona las réplicas usando la estrategia Least_Load.

Paso 2. Estado del Sistema Actual. En un tiempo "t", el estado del sistema actual es revisado. Por ejemplo, considerando el siguiente estado.

Réplica 1 = 7(4)
Réplica 2 = 8(6)
Réplica 3 = 9(3)
Réplica 4 = 10(6)

Paso 3. Nuevos Requerimientos Serán Planeados. Se toma de un pool todos los requerimientos que han arribado en grupos de demandas consecutivas.

11(9) 12(8) 13(7) 14(9) 15(17) 16(18) 17(4) 18(11) 19(17) 20(5)

Cada demanda expresa un grupo de requerimientos que solicitan servicios.

Réplica	Tarea Actual	Cola del Procesador	Carga Total en la Réplica
1	7(4)	15(17) 17(4) 20(5)	30
2	8(6)	16(18) 13(7)	31
3	9(3)	19(17) 18(11)	31
4	10(6)	12(8) 14(9) 11(9)	32
Carga Total en el Sistema			124

Tabla 3.1 Información de carga en un tiempo t+1.

Paso 4. Una Estrategia de Optimización Heurística es Aplicada. Basado en el conjunto de tareas del paso 2 y 3.

Paso 5. Asignación de Tareas. Después de "N" procesos la estrategia de optimización determinó que la mejor planeación es la que muestra la Tabla 3.1.

Paso 6. Un Nuevo Estado del Sistema. Nuevos requerimientos de los clientes serán planeados y el proceso se repite en los pasos 2 a 5.

Plataforma de Hardware

Actualmente el sistema de optimización cuenta con 9 PC´s dedicadas a los clientes, tienen sistema operativo Windows XP Profesional, 2GB en RAM y son AMD Athlon 64 X2 Dual Core Processor. Para las réplicas, se cuenta con 4computadoras, 2 mantienen las mismas características que sus clientes, 2 réplicas con sistema operativo Fedora Linux versión 12.0, Intel Core 2 Duo, CPU 2.93 GHz. El analizador de carga y el monitor son PC´s que mantienen las mismas características que los clientes. Todas las computadoras están conectadas en una red LAN Base 100 utilizando un Switch CISCO de 24 puertos, todas corren bajo el estándar de CORBA implementado en JAVA-IDL de SUN Microsystems versión 1.6 y posteriores (Figura 3.2).

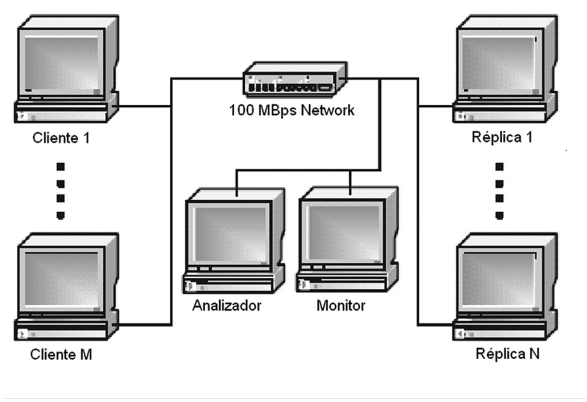

Figura 3.2 Plataforma de Hardware del Sistema.

Análisis Comparativo y Resultados

A continuación mostramos los resultados que arrojó el comparativo de las estrategias heurísticas en conjunto con LL. Las estrategias se implementaron bajo CORBA utilizando JAVA-IDL.

Se consideraronlos siguientes índices de desempeño [16]:

- Tiempo de Respuesta Global
- Rendimiento del Sistema (Throughput)

Los índices son tomados como métricas de comparación, otros factores tales como: políticas de migración, políticas de transferencia, políticas de carga, etc., también pueden aplicarse y se pueden consultar en [16].

Optimización Dinámica de Carga

Las estrategias se realizaron de la siguiente manera:

Primero. Aplicamos la optimización dinámica de carga utilizando diferentes tipos de requerimientos de clientes, arquitecturas y velocidades de procesamiento.

Segundo. Implementamos una aplicación cuyo objetivo fue la transferencia de archivos.

Número de Archivos	LL + Evolutiva	LL+ Tabú	LL + SA
4500	0.444	0.501	0.515
9000	1.401	1.991	2.004
18000	3.990	4.100	4.973
27000	4.533	4.994	5.001
36000	5.559	5.992	6.075
45000	7.002	7.918	8.006

Tabla 4.1 Tiempo de Respuesta Total (Granularidad=10 bytes).

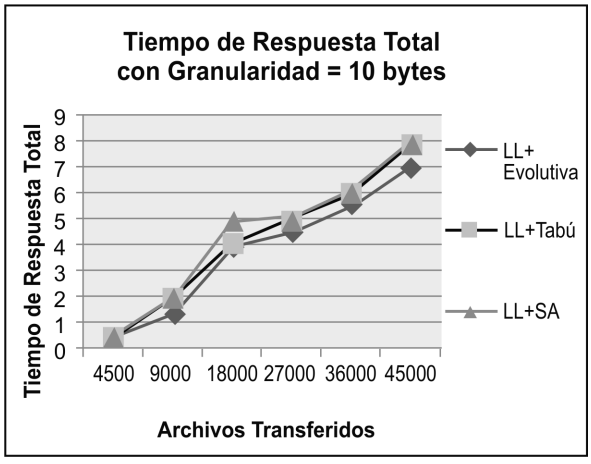

Figura 4.1. Tiempo de Respuesta Total (Granularidad = 10bytes).

Tercero. Para la evaluación utilizamos archivos de 4500, 9000, 18000, 27000, 36000 y 45000 archivos con granularidad constante (peso). La Figura4.1 muestra el tiempo de respuesta total de los diferentes algoritmos cuando el peso del archivo fue de 10 bytes.

Cuarto. Tomamos como base el número de archivos del punto anterior y elevamos su peso a 100kb, una vez elevado el peso del archivo este se mantuvo constante. En la Figura 4.2 mostramos el tiempo de respuesta total, cuando la granularidad fue igual a 100kbytes.

Número de Archivos	LL + Evolutiva	LL+ Tabú	LL + SA
4500	1.348	2.004	2.786
9000	2.676	3.076	3.700
18000	4.719	5.619	5.990
27000	6.688	7.463	7.999
36000	8.753	9.410	10.500
45000	9.007	10.484	11.003

Tabla 4.2. Tiempo de Respuesta Total (Granularidad=100kbytes).

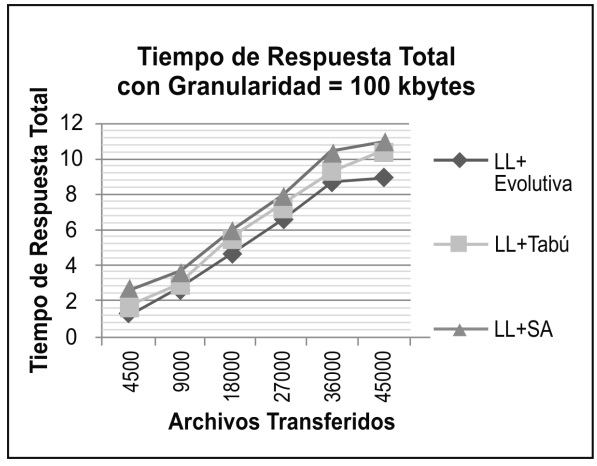

Figura 4.2. Tiempo de Respuesta Total (Granularidad = 100kbytes).

Quinto. Consideramos nuevamente el número de archivos de los dos puntos anteriores, pero ahora usamos archivos con formato MP3. La Figura 4.3 muestra el tiempo de respuesta total cuando la granularidad fueron archivos con formato MP3.

Número de Archivos	LL + Evolutiva	LL+ Tabú	LL + SA
4500	5.053	5.025	5.060
9000	8.002	8.682	8.701
18000	11.065	12.029	13.001
27000	14.075	15.045	16.044
36000	18.008	19.020	21.999
45000	22.025	23.690	24.695

Tabla 4.3. Tiempo de Respuesta Total (Archivos MP3).

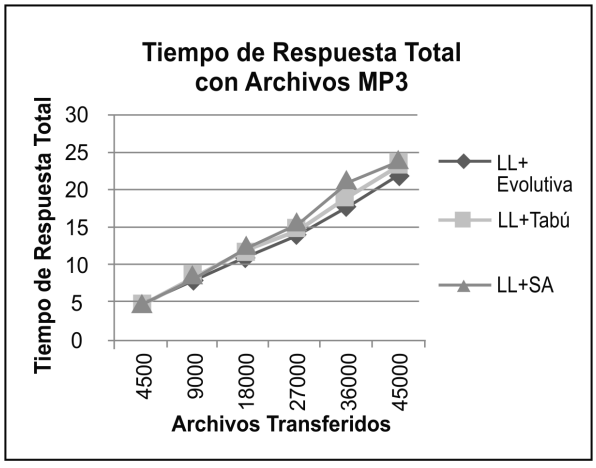

Figura 4.3. Tiempo de Respuesta Total (Archivos MP3).

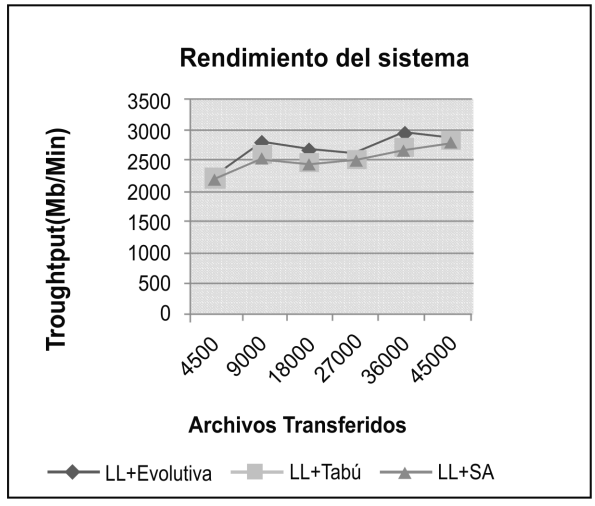

Figura 4.4. Optimización de Recursos con Archivos MP3 (Mb/Min).

Sexto.La utilización de recursos (throughput) se puede ver reflejada en la Figura 4.4, donde la estrategia LL + Evolutiva fue la que mejor aprovechó los recursos y esto es obvio, ya que esta estrategia es la que mejores tiempos de respuesta genera. En Tabla 4.4mostramos el throughput medido en base a la cantidad de megabytes tranferidos por minuto (Mb/Min) por replica, no la cantidad global de archivos o de megabytes transferidos por el sistema.

Apoyándose en los resultados que reflejaron las gráficas anteriores, pudimoselegir a LL +Evolutiva como la estrategia más adecuada para mejorar la estrategia LL.

Rendimiento del Sistema(Mb/Min)			
Número de Archivos	LL + Evolutiva	LL+ Tabú	LL + SA
4500	2222.7	2211.115	2201.705
9000	2811.705	2593.02	2553.02
18000	2706.075	2496.04	2450.5
27000	2620.36	2531.4	2504.995
36000	2976.715	2723.75	2659.25
45000	2869.515	2834.865	2800.005

Tabla 4.4. Optimización de Recursos con Archivos MP3(Mb/Min).

Conclusiones y Trabajo a Futuro

El objetivo del trabajo descrito anteriormente fue el de mejorar la estrategia dinámica LL propuesta por IONA Technologies. En este caso combinamos la estrategia dinámica LL con una estrategia heurística con el propósito de mejorar el tiempo de respuesta global. La LL en conjunto con la estrategia heurística se implementó en una arquitectura para optimizar carga bajo el estándar de CORBA.

Como pudo observarse, en el análisis comparativo evaluamos las estrategias utilizando diferentes métricas de carga e ilustramos como la estrategia LL + la Evolutiva, siempre generó las mejores condiciones de optimización de carga, por lo que nos llevó a concluir, que la estrategia Evolutiva en conjunto con LL es una buena opción para mejorar el tiempo de respuesta global de LL.

Una vez detectada la estrategia heurística (Evolutiva) que mejora LL, pretendemos evaluar en un futuro aplicaciones que involucren gran tiempo de procesamiento, por ejemplo, en el reconocimiento y verificación de firmas manuscritas On-Line y Off-Line (en línea y fuera de línea) en donde el tiempo de respuesta es sumamente demandante.

Referencias

[1] George Coulouris, Jean Dollimore , Tim Kindberg, and Gordon Blair. (2011) *Distributed Systems: Concepts and Design*. 5th Edition . Addison Wesley.

[2] James Little and Peter Membrey. (2012). *Practical Load Balancing: Ride the Performance Tiger***.** First Edition, Apress 2012.

[3]. Catalog of OMG CORBA®/IIOP® Specifications 2012. http://www.omg.org/technology/documents/corba_spec_catalog.htm

[4] Luna Fco. Javier, Martínez Julio Cesar (2007), "Improving Dynamic Load Balancing Under CORBA With a Genetic Strategy in a Neural System of Off-line Signature Verification". The 2007 International Conference on Parallel and Distributed Processing Techniques and Applications. In *Computer Science & Computer Engineering*, Las Vegas Nevada, USA.

[5] Luna Francisco, Martínez Julio, Medina Gricelda, Mora Miguel, López Valentín (2010), Optimizing the Total Execution Time from a Neural System of Off-line Signature Verification*. CIIIE Aguascalientes Mex. IEEE Sección Aguascalientes.*

[6] Luna Fco. Javier, Alcántara Silva Rogelio (2005). Combining Genetic Strategy with Least-Loaded to Improve Dynamic Load Balancing in CORBA. The 2005 International Conference on Paralled and Distributed Processing Techniques and Applications. In *Computer Science & Computer Engineering,* Las Vegas Nevada, USA, ISBN: 1-60132-093-0, 1-60132-094-9 (1-60132-095-7) CSREA.

[7] IONA Technologies, Tri-Pacific Software Inc. and VERTEL Corporation (2002), *Load Balancing and Monitoring*. Revised Joint Submission (mars/02-04-05).

[8] Booch Grady, Rumbaugh James y Jacobson Ivar (1999), *El Lenguaje Unificado de Modelado*. Addison Wesley.

[9] Booch Grady, Rumbaugh James y Jacobson Ivar (1999), *The Unified Software Development Process.* Addison Wesley.

[10] Luna Fco. Javier, Martínez Romo Julio Cesar, Medina Veloz Gricelda (Septiembre 2008). Optimizando el Balanceo Dinámico de Carga bajo CORBA en un Sistema Neuronal de Verificación de Firmas off-line. Revista Indizada, *Investigación y Ciencia*-UAA ISSN= 1665-4412 Volumen No. 41.

[11] Medina Veloz Gricelda, Fco. Javier Luna Rosas (Septiembre 2008). Integrando Mediante Patrones de Software una Estrategia Fuzzy-Logic, en un Servicio de Balanceo Dinámico de Carga bajo CORBA. Revista Indizada, *CONCIENCIA TECNOLOGICA* ISSN=1405-5597 Volumen No. 35.

[12] Fowler Martin con Scott Kendall (1997), UML Gota a Gota. *Addison Wesley.*

[13] Buschman Frank, et al. (1996) *A system of patterns, Pattern-Oriented, Software-Architecture*. John Wiley and Sons Ltd.

[14] Gamma, E., et al. (2002), Patrones de diseño: Elementos de software Reutilizable*, Addison Wesley.*

[15] Ossama Othman, O´Ryan Carlos and Schmidt C (2001), Designing an Adaptive CORBA Load Balancing Service Using TAO. *"IEEE Distributed Systems on Line*", 2.

[16] Goscinski, Andrzej (1992), Distributed Operating Systems. The Logical Design*. Addison-Wesley.*

[17] El-Ghazali Talbi (2009). *Metaheuristics: From Design to Implementation*. Wiley Series on Parallel and Distributed Computing.

[18] GüntherZäpfel (2010). *Metaheuristic Search Concepts: A Tutorial with Applications to Production and Logistics.* Springer-Verlag Berlin Heidelberg.

[19] Xin-She Yang (2010). *Engineering Optimization: An Introduction with Metaheuristic Applications*. Wiley.

[20] Fernandes C.M., Laredo J.L.J., Mora A. M., Rosa A.C. and Merelo J.J (December 9, 2011). The Sandpile Mutation Operator for Genetic Algorithms. Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011, Lecture Notes in Computer Science and General Issues, ISBN-10: 3642255655, ISBN-13: 978-3642255656.

[21] Ghada F., EL-Kabbany, Nayer M. Wanas, Nadia H. Hegazi, Samir I. Shaheen (2011). Dynamic Load Balancing Framework for Real-time Applications in Message Passing Systems. Journal: Simulation Modelling Practice and Theory- SIMUL MODEL PRACT THEORY, vol. 19, no. 4, pp. 1021-1034.

[22] The current official version of UML and its associated specifications 2012. http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML

Recibido: 15 de septiembre de 2011
Aceptado: 9 de julio de 2012