Conciencia Tecnológica No. 35, Enero - Junio 2008

Integrando Mediante Patrones de Software una Estrategia Fuzzy-Logic, en un Servicio de Balanceo Dinámico de Carga bajo CORBA

Investigación

Lic. Gricelda Medina Veloz[2], Dr. Francisco Javier Luna Rosas[1], Dr. Jaime Muñoz Arteaga[2], Dr. Julio César Martínez Romo[1]

[1] Instituto Tecnologico de Aguascalientes, Av. Adolfo López Mateos 1801 Ote. Esq. Av. Tecnológico, Fracc. Bona Gens, C.P. 20256, Aguascalientes, Ags. e-mail: fjluna@ita.mx, jucemaro@yahoo.com

[2] Universidad Autónoma de Aguascalientes, Av. Universidad 940, Aguascalientes Ags., C.P. 20100, e-mail: grismv2000@yahoo.com , jmunozar@correo.uaa.mx

Resumen

El enfoque del uso de patrones en el desarrollo de software se ha popularizado a nivel mundial y dentro de la ingeniería de software se ha extendido cada vez más su uso, a tal grado que se han creado y se siguen creando catálogos de patrones para solucionar problemas en el desarrollo de software a distintos niveles de abstracción. Este trabajo se centra en destacar el diseño mediante patrones de software de un servicio de balanceo de carga para aplicaciones distribuidas desarrolladas bajo el estándar de CORBA. En el servicio de balanceo se implementan patrones de software de diseño y de arquitectura. Para brindar el balanceo se propone implementar una nueva estrategia definida mediante lógica difusa con el propósito de mejorar el balanceo de carga en un sistema distribuido a gran escala.

Palabras clave

CORBA, balanceo de carga, Fuzzy Logic, patrones de software.

Abstract

The approach of use of patterns in the software development has become popular world wide and within the software engineering its use has been so widespread to the extent that pattern cataloges have been created to solve problems about software development at different abstraction levels. This work is centered in highlighting the design through software patterns of a load balancing service for distributed applications under CORBA standard. In the load balancing service, architecture and design software patterns are implemented. In order to provide the balancing it is proposed to implement a new defined strategy through fuzzy logic with the purpose of improving the load balancing on a distributed system at great scale.

Keywords

CORBA, load balancing, Fuzzy Logic, software patterns.

Introducción

En este documento se toma la idea de reutilización de software y facilidad de uso que ofrecen los patrones a su desarrollo para diseñar y construir un servicio de balanceo de carga para aplicaciones distribuidas bajo CORBA, además de integrar también una estrategia de balanceo adicional a las propuestas por la literatura, basada en la lógica difusa, con el objetivo de mejorar el nivel de balanceo de carga en aplicaciones de sistemas distribuidos a gran escala.

El resto del documento está organizando de la siguiente manera: En la sección dos se definen los conceptos claves para un servicio de balanceo basado en el estándar de CORBA, en la tres se presenta la problemática a la cuál se hace referencia con el diseño del servicio de balanceo, la sección cuatro describe a grandes rasgos la propuesta para la mejora al servicio de balanceo en sistemas a gran escala, y en la cinco se describe de manera detallada los patrones de software a implementar en su diseño y construcción. La sección seis define la estrategia de balanceo propuesta basada en el modelo de lógica difusa, sus componentes, reglas y operadores. En la sección siete se mencionan algunos trabajos relacionados de otros servicios de balanceo sobre el mismo estándar, así como patrones de software aplicados a arquitecturas de balanceo generales. Finalmente la sección ocho define los

alcances y limitaciones del proyecto así como las conclusiones del mismo.

Un servicio de balanceo bajo CORBA

Los conceptos claves de un servicio de balanceo de carga basado en CORBA son mostrados en la Figura 1 y son descritos como sigue [15], [1]:

Balanceador de carga. Es un componente que intenta distribuir la carga a través de grupos de servidores de una manera óptima. Un balanceador de carga debe consistir de un simple servidor centralizado o múltiples servidores descentralizados que colectivamente forman un balanceador lógico.

Réplica. Es un duplicado de un objeto particular sobre un servidor que es manejado por un balanceador de carga. Ésta ejecuta las mismas tareas que el objeto original.

Grupo de objetos. Es un grupo de réplicas a través del cual la carga es balanceada. Las réplicas en tal grupo implementan las mismas operaciones remotas.

Sesión. En el contexto de balanceo de carga middleware, una sesión se define como el periodo de tiempo que un cliente invoca operaciones remotas (requerimientos) para acceder servicios proporcionados por objetos en un servidor particular (Figura 1).

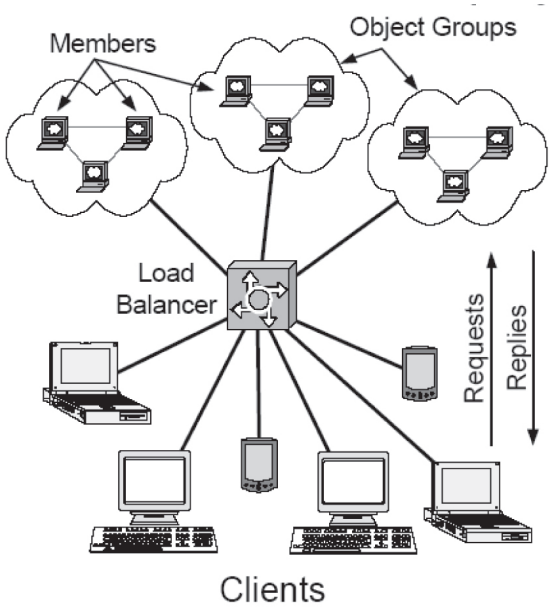

Figura 1. Conceptos Claves en un Servicio de Balanceo de Carga Middleware [10].

Problemática

El surgimiento de la computación de objetos distribuidos representa la confluencia de las dos mayores tecnologías de información, los RPC's y la programación y diseño orientados a objetos. CORBA y Java RMI son los ejemplos más representativos de esta tecnología [9].

Ha habido importantes enfoques para extender funcionalidades de CORBA para que soporte balanceo de carga, por ejemplo, TAO [16], VisiBroker [11], MICO [17], [13], etc.

Para consolidar este enfoque y resolver el problema, la OMG diseño un Request For Proposal (RFP) [14], que es una propuesta para extender la funcionalidad de CORBA para balancear y monitorear carga en ambientes basados en CORBA; para procesar distribuidamente y con alto desempeño las aplicaciones implementadas en el estándar.

De los trabajos recibidos, el artículo de IONA Technologies [10] fue el trabajo más relevante en esta área y la OMG lo ha recomendado para su adopción. El artículo incluye tres estrategias que deberán soportar las implementaciones CORBA: dos estrategias estáticas (Round-Robin y Random) más una estrategia dinámica (Least-Loaded (LL)).

En las estrategias estáticas el atado de los requerimientos y la política son aplicadas por toda la vida del cliente.

En las estrategias dinámicas se utiliza 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 ésta no proporcione el resultado esperado por el sistema.

El hecho de que esta tecnología ofrezca una serie de servicios en forma de objetos, hace que el diseño y desarrollo de sus aplicaciones se realice basándose en interfaces bien definidas que faciliten y apoyen la modularidad y la reutilización, a la vez que permiten un diseño mucho más flexible y eficiente, haciendo esto que la complejidad inherente al diseño y desarrollo de los sistemas distribuidos sea muy alta [4], provocando que este tipo de sistemas sean más difíciles de diseñar, depurar y mantener que otras piezas de software. Esto implica especificar innumerables detalles para que su desarrollo sea eficiente y rápido. Para lograrlo

se necesitan soluciones formales claras, efectivas y comprensibles, aplicables en la gestión de proyectos de software, y los patrones pretenden ser la solución al problema de la comunicación de experiencias y conocimientos entre desarrolladores de software, facilitando la reutilización del diseño y la arquitectura del software, además de permitir la reestructuración de los componentes generados por los sistemas [8], [6], [4].

Mejora propuesta al servicio de balanceo

Los desafíos que surgen con la construcción del los sistemas distribuidos son la heterogeneidad de sus componentes, su entorno abierto para permitir añadir o quitarlos, la seguridad y la escalabilidad que les permiten funcionar adecuadamente al incrementar el numero de usuarios [05].

Mediante la técnica de balanceo de carga o también denominada planificación de tareas se propone mejorar la escalabilidad de este tipo de sistemas, además de ayudar a que los recursos del sistema sean aprovechados más eficientemente [15].

Por otro lado, a través del enfoque del uso de patrones, por su naturaleza genérica de Problema-Solución-Reutilización [8], se desea brindar a este tipo de sistemas desde su etapa de diseño la capacidad de adaptación y posible extensión a su comportamiento.

También se propone agregar al servicio de balanceo una estrategia adicional, a las propuestas por la literatura, definida mediante un modelo de lógica difusa, con la cual se ofrece la posibilidad de balancear aplicaciones, en sistemas distribuidos a gran escala, construidas bajo el estándar de CORBA.

Patrones de software en el servicio de balanceo

Patrón Broker

Ayuda a resolver los problemas derivados del diseño de sistemas distribuidos y heterogéneos, referentes a la distribución de la infraestructura del software, simplificando las aplicaciones de sistemas distribuidos, y la comunicación fundamental soportados por productos y plataformas middleware [3], [4], Figura 2.
En el servicio de balanceo diseña y representa el equilibrio que permite a los componentes acceder a los servicios que ofrecen otros componentes mediante invocaciones de servicio remotas y transparentes a la localización de los servidores, para cambiar, añadir o eliminar componentes en tiempo de ejecución. En el balanceo estos requerimientos son pasados a través de proxies al broker. El broker busca para su localización al objeto servidor correspondiente y le pasa los requerimientos.

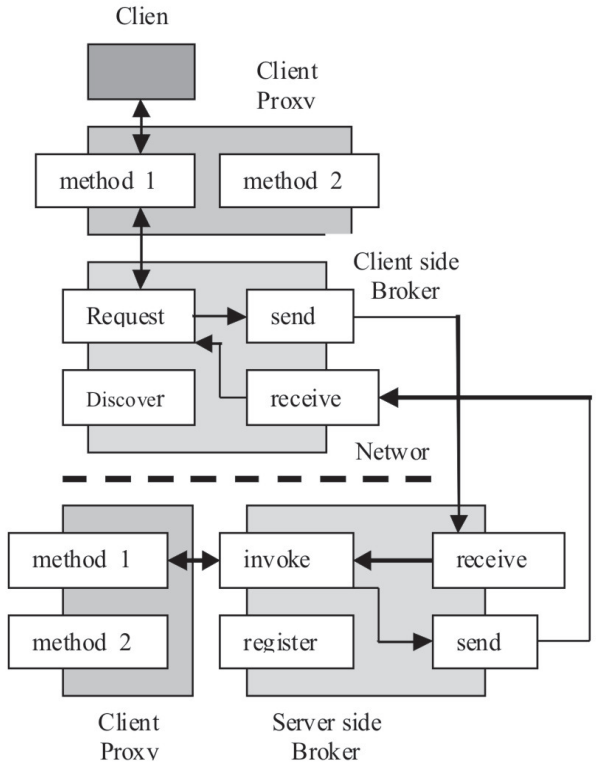

Figura 2. Patrón Broker

Patrón Interceptor

Permite agregar fácilmente funcionalidad al sistema para cambiar su comportamiento dinámicamente, sin necesidad de recompilarlo, es decir, hace el cambio de comportamiento en tiempo de ejecución [7], [19] ya sea interponiendo una nueva capa de procesamiento o cambiando el destino dinámicamente, ya que interpone objetos los cuales pueden interceptar llamadas e insertar un procesamiento específico que puede estar basado en el análisis del contenido, además de redireccionar una llamada a un punto diferente Figura 3 [4].

En el servicio de balanceo, este patrón representa el mecanismo para redirigir los requerimientos de los clientes a una réplica apropiada. Las aplicaciones basadas en CORBA contienen las construcciones necesarias para soportar transparentemente el envío de requerimientos de los clientes [19].

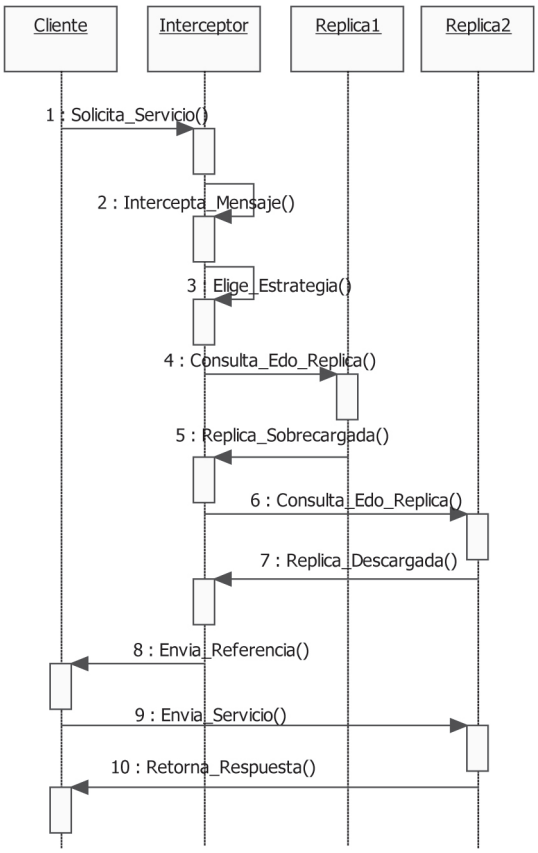

Figura 3. Diagrama de secuencia del patrón Interceptor.

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 balanceador de carga debe ser lo suficientemente flexible para soportar diferentes tipos de estrategias de balanceo, tanto estáticas como dinámicas [21], [3], (Figura 4).

Las estrategias encapsuladas en este patrón son las siguientes:

Random. Esta estrategia es no-adaptativa y selecciona aleatoriamente una réplica de un grupo de réplicas, para el envío de requerimientos.

Round Robin. Estrategia no adaptativa (no considera las condiciones de carga en tiempo de ejecución) y simplemente escoge una réplica de un grupo de réplicas para enviar los requerimientos del cliente, seleccionándola rotativamente de un grupo de objetos dado.

Least Loaded (LL). La estrategia LL es utilizada para el balanceo dinámico de carga, a diferencia de la estrategia Random y Round Robin, la estrategia LL usa una estrategia de balanceo de carga adaptativa. Como su nombre lo indica esta estrategia elige dinámicamente al miembro de un grupo de objetos con la carga más baja utilizando información en tiempo de ejecución.

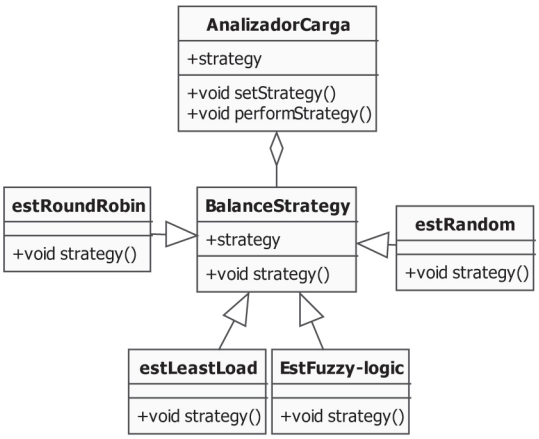

Figura 4. Diagrama de clases del patrón estrategia.

Mas la estrategia adicional propuesta basada en Fuzzy-Logic

En el servicio de balanceo propuesto, estas estrategias son encapsuladas en el componente analizador de carga, el analizador de carga usa el patrón estrategia para configurar el algoritmo de balanceo que se usará cuando se tomen las decisiones de balancear.

La propuesta primaria del analizador, es determinar las condiciones de carga de las réplicas y grupos de réplicas registradas, y dependiendo de la implementación, un soporte de múltiples estrategias de balanceo estará disponible. Dado que el conocimiento a priori de los requerimientos exhibidos por una aplicación distribuida no siempre está disponible, el patrón estrategia brinda la habilidad para dinámicamente agregar o remplazar las estrategias de balanceo soportadas por el analizador de carga dado.

La estrategia de balanceo Fuzzy-Logic

En publicaciones anteriores se realizó la optimización del tiempo de respuesta global de una aplicación aplicando una estrategia genética [12][13]. Los resultados fueron alentadores ya que la mejor forma de resolver un problema de balanceo de carga es centrar el problema como un problema de optimización. En la última década, los Algoritmos Genéticos (AGs) 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. Una explicación detallada acerca de algoritmos genéticos para optimizar el tiempo de respuesta de una aplicación en el balanceo dinámico de carga está publicada en [12] [13].

Sin embargo en estos trabajos el tiempo de respuesta del canal de comunicación no incide en el cálculo para la toma de decisiones del balanceo de carga. En la actualidad la mayoría de los servicios de balanceo generalmente se dan a nivel de Web, el tiempo de respuesta del canal si incide, ya que este es un canal abierto que involucra cierto grado de incertidumbre.

Un modelo Fuzzy-Logic generalmente es utilizado para cálculos en aplicaciones que involucran cierta incertidumbre, esta es la razón principal por la cual la estrategia propuesta para el balanceo carga, incluye el uso de la lógica difusa, con ello se pretende dar solución a la optimización de recursos en sistemas a gran escala combinando el tiempo de respuesta del canal con el tiempo de respuesta de las réplicas o servidores.

Un controlador fuzzy incluye 5 componentes, los cuales son: fusificación, base de reglas, funciones de membresía, máquina de inferencia fuzzy y defusificación, Figura 5 [22], [23].

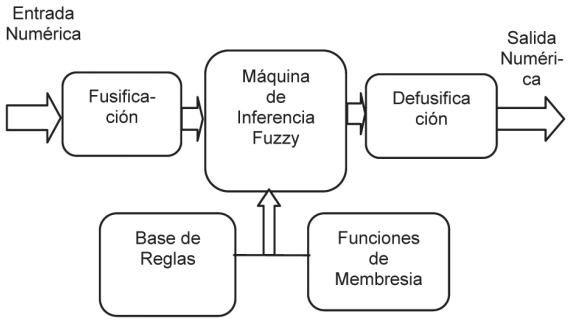

Figura 5. Arquitectura del Modelo Fuzzy-Logic.

Fusificación. Es la interfaz de entrada que mapea una entrada numérica a un conjunto fuzzy para que este pueda ser correspondido con las premisas de las reglas fuzzy definidas en la base de reglas especificas para la aplicación.

Base de Reglas. Contiene un conjunto de reglas de fusificación if-then que define las acciones del controlador en términos de variables lingüísticas y funciones de membresía de términos lingüísticos.

Funciones de Membresía. La función de membresía o pertenencia de un conjunto fuzzy consiste en un conjunto de pares ordenados si la variable es discreta, o una función continua si no lo es. El valor de indica el grado en que el valor de la variable está incluida en el concepto representado por la etiqueta. Para la definición de estas funciones de pertenencia se utilizan convencionalmente ciertas familias de forma estándar, por coincidir con el significado lingüístico de las etiquetas más utilizadas. Las más frecuentes son la función de tipo trapezoidal, triangular, exponencial, etc.

Máquina de Inferencia Fuzzy. Aplica el mecanismo de inferencia al conjunto de reglas en la base de reglas fuzzy para producir un conjunto de salida.

Defusificación. Es un mapeador de salida que convierte el conjunto fuzzificado de salida, a una salida crisp. Basada en la salida crisp, el controlador fuzzy puede manejar el sistema bajo cierto control.

Para hacer una decisión de balanceo correcta, las variables lingüísticas de la carga de las réplicas (LoadReplicas), el tiempo de invocación de métodos remotos (RMIT) y la calidad de servicio (QualityService) es definida en el modelo fuzzy de la siguiente manera:

Carga de las Réplicas

Se define la carga de la réplica (LoadReplica), con la definición del conjunto fuzzy: {Low, Medium, High}. Se emplea un enfoque indirecto para medir la carga de la Réplica, en lugar de medir directamente la carga de cada una, se mide la carga del grupo de réplicas (benchmark) que es un grupo de réplicas controladas por un manejador de carga. El benchmark esta siempre en ejecución como un proceso background. La Figura 6 muestra la gráfica de membresía para la carga de la réplica.

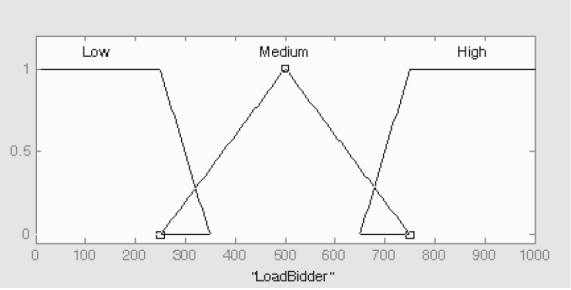

Figura 6. Gráfica de Membresía para la Carga de las Réplicas.

Tiempo del método de invocación remoto (RMIT)

La forma de medir la respuesta de las réplicas al manejador así como el overhead introducido, es calculando la utilización de la red. Esto se hace midiendo el tiempo de invocación de métodos remotos (RMIT), Figura 7.

Cabe mencionar que se está monitoreando el canal de CORBA y dentro de éste se deben considerar los siguientes cuatro puntos en el monitoreo: Send Request, Receive Request, Send Reply and Receive Reply [18].

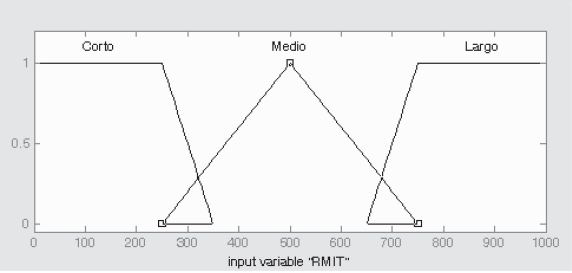

Figura 7. Gráfica de Membresía para el Tiempo de Respuesta Remoto (RMIT).

Estos cuatro puntos están inmersos en la llamada al método RMI y se define este método simplemente retornando un tipo de datos primitivo de la réplica al manager midiendo el tiempo. El método System. currentTime() es usado para medir el tiempo de enlace durante la invocación del método remoto, y esta calculado en milliseconds. El conjunto fuzzy para el Remote Method Invocation Time (RMIT) esta definido como: {Short, Medium, Long}. La figura anterior muestra la gráfica de membresía para medir el tiempo de invocación del método remoto (RMIT).

Calidad del servicio (QualityService)

Se usa QualityService para clasificar servicios en seis diferentes categorías: {VeryLow, Low, MédiumLow, Médium, MediumHigh, High}.

Figura 8. Gráfica de Membresía para la Calidad del Servicio (QualityService).

Después de definir las variables fuzzy se define el conjunto de reglas de inferencia como sigue:

- 1. **If** (LoadReplica is Low) **and** (RMIT is Short) **then** (QualityService is High)
- 2. **If** (LoadReplica is Low) **and** (RMIT is Medium) **then** (QualityService is MediumHigh)
- 3. **If** (LoadReplica is Low) **and** (RMIT is Long) **then** (QualityService is Medium)
- 4**. If** (LoadReplica is Medium) **and** (RMIT is Short) **then** (QualityService is MediumHigh)
- 5. **If** (LoadReplica is Medium) **and** (RMIT is Medium) **then** (QualityService is Medium)
- 6. **If** (LoadReplica is Medium) **and** (RMIT is Long) **then** (QualityService is Low)
- 7. **If** (LoadReplica is High) **and** (RMIT is Short) **then** (QualityService is MediumLow)
- 8. **If** (LoadReplica is High) **and** (RMIT is Medium) **then** (QualityService is Low)
- 9. **If** (LoadReplica is High) **and** (RMIT is Long) **then** (QualityService is VeryLow)

La gráfica de membresía es mostrada en la Fig. 8.

Resultados del modelo Fuzzy

Para aplicar las reglas de inferencia, una decisión puede ser generada basada en ambos antecedentes. Esto es, si RMIT es short y LoadReplica es Low, entonces QualityService es High (ver Figura 9).

Teniendo esas reglas de inferencia y las gráficas de membresía, el proceso de fusificación y defusificación puede ser llevado a cabo como sigue: Primero las variables de entrada de RRMIT y LoadReplica son mapeadas a sus respectivos valores de membresía de acuerdo a las gráficas de membresía esos valores son comparados y el mínimo de los dos es entonces proyectado sobre la función de membresía de su gráfica consecuente. Después de que la gráfica de salida es generada, la defusificación de la salida fuzzy dentro de un crisp o valor numérico puede ser llevada a cabo. Se usa el método del centroide para traslapar las áreas [22], [23].

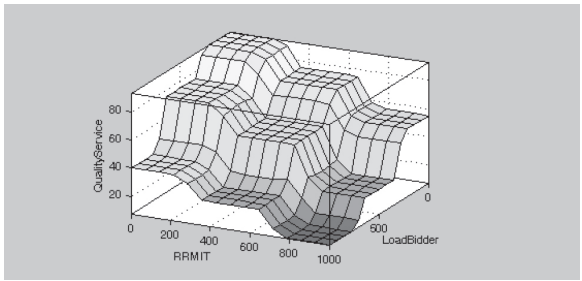

Figura 9. Gráfica de Salida para QualityService.

El centroide global de las áreas traslapadas A_i para $i = 1, 2, ..., N$ está dado por:

$$\overline{X} = \frac{\sum_{i=1}^{N} \overline{x}_i A_i}{\sum_{i=1}^{N} A_i} \quad \text{.....(5.1)}$$

Donde A_i y $\overline{x}_i$ son el área traslapada y el centroide de los triángulos (triangles) o trapecios (trapeziums) obtenidos en la i^{th} regla. El centroide y el área son calculados para cada triángulo o trapecio. Este proceso es repetido para otras reglas de inferencia donde las entradas son aplicadas para obtener un área compuesta de trapecios traslapados. El proceso de defusificación genera un valor centroide que representa el rango de la QualityService. El Manager puede entonces enviar los requerimientos del cliente a la mejor Réplica basándose en la QualityService obtenida para cada Réplica. En la Figura 8 se observa la QualityService si se tienen diferentes entradas que representan la carga en las Réplicas y el tiempo de respuesta remoto que genera cada uno en la red.

La Tabla 1 representa 20 entradas simuladas en el modelo fuzzy, las entradas tanto para la carga como para los tiempos son generadas de manera aleatoria.

En la entrada uno se observa claramente que cuando las Réplicas manejan una carga elevada y el tiempo de respuesta de la red se encuentra en un tiempo medio, el sistema infiere que QualityService no es muy bueno (25%).

Number	Load Bidder	RRMIT(MilliSeconds)	Quality Of Service (0-100%)
1	950	583	25
2	231	423	75
3	607	516	55
4	486	334	60
5	891	433	25
6	762	226	40
7	456	580	55
8	19	760	55
9	821	530	25
10	445	641	55
11	615	209	75
12	792	380	25
13	922	783	8
14	738	681	22
15	176	461	75
16	406	568	55
17	935	794	8
18	917	59	40
19	410	603	55
20	894	50	40

Tabla 1. Carga de las Réplicas (LoadReplica), Tiempo de Respuesta en la Red (RMIT) y Calidad del Servicio (QualityService).

En la segunda entrada las Réplicas tienen baja carga, pero siguen manteniendo la respuesta de la red en un tiempo medio, por lo tanto el sistema infiere que la QualityService es medio alta (75%). En la entrada 13 y 17 cuando las Réplicas tienen un tiempo de respuesta alto y una carga elevada, el sistema infiere que el QualityService es muy malo (8%). La gráfica 10 muestra el QualityService que genera el modelo fuzzy cuando se simulan 100 entradas de las Réplicas.

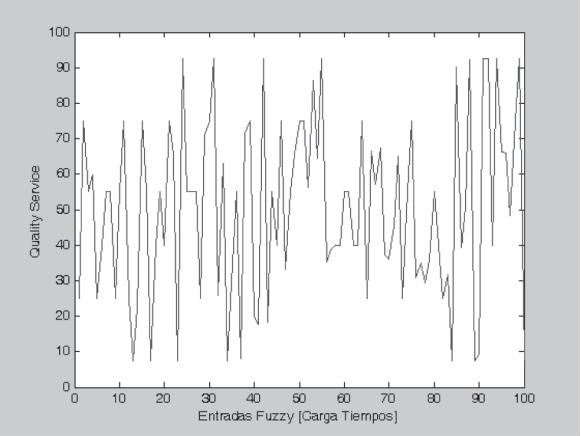

Figura 10. Quality of Service de las Réplicas.

Trabajos relacionados

Existen diseños importantes de patrones respecto a métodos de balanceo de carga que ayudan a construir de una forma más rápida y eficiente el diseño de las estructuras y mecanismos de colaboración entre los componentes que integran la arquitectura [15], [1], [20].

En esta sección se mencionan algunos patrones de diseño importantes en las arquitecturas de balanceo.

Patrón de Grupo de Objetos. El patrón de diseño de grupos de objetos describe la implementación de replicación de objetos. El cliente tiene una referencia de objetos CORBA que representa varias réplicas. Esto significa que cuando el cliente diseña un requerimiento a una referencia de grupo de objetos, este requerimiento es enviado a las réplicas [1].

Patrón Proxy. El patrón obliga a que las llamadas a métodos de un objeto ocurran indirectamente a través de un objeto proxy, que actúa como sustituto del objeto original, delegando luego las llamadas a los métodos de los objetos respectivos [4].

En CORBA los proxies son usados del lado del cliente, como objetos de lenguajes de programación que ofrecen las interfaces (de acuerdo al mapeo del lenguaje IDL del lado del cliente dado) para objetos CORBA representados. Cuando el cliente es invocado, este es responsable de enviar las invocaciones al servant actual via ORB.

Patrón cliente despachador [3]. Corresponde directamente a un componente naming/trading. Los clientes preguntan al despachador por la referencia

de objetos que serán usados para subsecuentes requerimientos, por lo que este patrón puede ser aplicado para realizar la asignación de requerimientos dinámicos basados en un trading.

También existen aportaciones importantes de arquitecturas de balanceo basadas en CORBA, entre las cuales se encuentran, Tabla 2:

Trabajo/ Característica		**LUNA [12]**	**TAO [16]**	**Trabajo actual**
Lenguaje	C++	JAVA-IDL	C++	JAVA-IDL
Tipo Red	LAN	LAN	LAN	WAN
Diseño Patrones	No	No	Si	Si
Estrategias de balanceo	Random Round-Robin LL	Random LL Round Robin Genetica		Random LL, Round Robin Fuzzy-Logic

Tabla 2. Trabajos relacionados

El trabajo de IONA Technologies [10], es considerado el trabajo más relevante por la OMG y lo ha recomendado para su adopción en respuesta a su Request For Proposal [14]. El trabajo de Luna, es una arquitectura de balanceo con una estrategia genética en un sistema neuronal, desarrollado en JAVA-IDL [12]. TAO de Cygnus es un servicio de balanceo y monitoreo para aplicaciones de tiempo real implementado en lenguaje C++ [16]. Borland VisiBroker es una herramienta para plataformas Microsoft que proporciona funcionalidad y balanceo para la interoperabilidad de aplicaciones CORBA con otras tecnologías como Web Services, .NET y J2EE [2].

Conclusiones y Trabajo Futuro

El modelado es la parte central de todas las actividades que conducen a la producción de un buen software. En este artículo se construyeron modelos mediante el uso de patrones de software, para comunicar la estructura y el comportamiento del sistema, para visualizar y controlar el servicio de balanceo, con el objetivo de cubrir las diferentes vistas y flujos de información a un nivel de abstracción más alto que el representado por las clases y los objetos, que ayuden a generar los planos para la automatización directa del balanceo de carga.

Además se propuso una nueva estrategia de balanceo para sistemas distribuidos a gran escala basada en un modelo de lógica difusa, la cual permite una mayor precisión al momento de la toma de decisiones en el manejo de sistemas con cierto grado de incertidumbre.

El sistema de balanceo será probado y comparado con algoritmos ajustados a demandas adaptativas, con la finalidad de mejorar el tiempo de respuesta global en el procesamiento de objetos distribuidos.

Para su evaluación serán consideradas aplicaciones de procesamiento distribuido de imágenes utilizando diversas operaciones como escala de grises, inverso o negativo, binarización, función de potencia, detección de bordes, filtros, etc.

Referencias

[1] Balasubramanian Jaiganesh, Schmidt Douglas C., Dowdy Lawrence, Othman Ossama. (2004) "Evaluating the Performance of Middleware Load Balancing Strategies," edoc, pp. 135-146, *Enterprise Distributed Object Computing Conference, Eighth IEEE International* (EDOC'04)

[2] BORLAND VisiBroker 7.0 Comprehensive ORBA Environment 2006, en línea consulta Mayo 2007 www.borland.com

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

[4] Buschman Frank, Henney Kevlin, C. Schmidt Douglas. (2007) *Pattern Oriented Software Architecture* "A pattern Language for Distributed Computing". Villey, Volume 4.

[5] George Colouris, Jean Dollimore, Tim Kindberg. (2005) *Sistemas distribuidos conceptos y diseño*, tercera Ed. Addison Wesley

[6] James O. Coplien, (2000), *Software Patterns*, Bell Laboratories, The Hillside Group.

[7] Curry E., D. Chambers, and G. Lyons, (2004) "Extending Message-Oriented Middleware Using Interception," *Proc. 3rd Int'l Workshop on Distributed Event-Based Systems* (DEBS 04), Institute of Eng. and Tech., pp. 32–37.

[8] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. (1995) *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley .

[9] Gopalan Suresh Raj. (September '98). A Detailed Comparison of CORBA, DCOM, and Java/RMI (with detailed code examples) Object Management Group (OMG) whitepaper.

[10] IONA Technologies, Tri-Pacific Software Inc, VERTEL Corporation, Load Balancing and Monitoring, Supported by Alcatel, Institute National des Telecommunications, Lucent Technologies, University of California – Irvine,

University of Toronto, Tech. Rep. In Response to OMG TC Document Orbos/2001-04-27, April 1, 2002

[11] Lindermeier M. (2000), Load Management for Distributed Object-Oriented Enviroments. In *2nd International Symposium on Distributed Objects and Applications* (DOA 2000), Antwerp, Belguim, Sept. OMG.

[12] Luna Rosas Fco. Javier, Alcántar Silva Rogelio. (June 27-30, 2005). Combining Genetic Strategy with Least-Loaded to Improve Dynamic Load Balancing in CORBA. *The 2005 International Conference on Parallel and Distributed Processing Techniques and Applications. In Computer Science & Computer Engineering*, Las Vegas Nevada, USA.

[13] Luna Fco. Javier, Martinez Julio César., (June, 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.

[14] Object Management Group, (Apr. 2001). Load Balancing and Monitoring for CORBA-based Applications, Request for Proposal, Tech. Rep., OMG Document (orbos/2001-04-27).

[15] Ossama Othman, O´Ryan Carlos and Schmidt. (March 2001). Strategies for CORBA Middleware-Based Load Balancing, "*IEEE Distributed Systems on Line*", Vol. 2, Number 3.

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

[17] Puder A. and Roemer K. (2002). *MICO: An Open Source CORBA Implementation*. Morgan Kaufmann, 3rd edition, Mar. 2002. ISBN:1558606661.

[18] Scallan Todd, (June-2000). Monitoring and Diagnostics of CORBA Systems. Demystifying the CORBA Communication Bus to Enable ´Distributed Debugging´. *Java Developers Journal*.

[19] Schmith Douglas, Stal Michael, Rohnert Hans, Buschman Frank. (2000) Pattern-Oriented Software Architecture Volume 2 – Patterns for Concurrent and Networked Objects. Chichester: Wiley

[20] Schnekenburger Thomas, (Feb, 2000). Load Balancing in CORBA: A Survey of Concepts, Patterns, and Techniques. *The Journal of Supercomputing*, Vol. 15 Num. 2, Pag. 141-161,

[21] Stephen Stelting, Olav Maasen. (2003) *Patrones de diseño aplicados a JAVA*. Primera edición en español, Pearson Educación, S.A, Prentice Hall Sun Microsystems.

[22] Yen John, Langari Reza. (1999). *FUZZY LOGIC, Intelligence, Control and Information*. Prentice Hall.

[23] Yu-Kwong Kwok, Lap-Sun Cheung. (2004) A new fuzzy-decision based load balancing system for distributed object computing. *Journal of Parallel and Distributed Computing* 64, 238-253.

Artículo recibido: 17 de enero de 2008

Aceptado para publicación: 19 de junio de 2008