Conciencia Tecnológica No. 32, Julio-Diciembre 2006

Combinando un Modelo-Fuzzy con el Algoritmo Request for Bids para Mejorar Least_Loaded en el Balanceo Dinámico de Carga Bajo CORBA en un Sistema Distribuido de Gran Escala

Investigación

Fco. Javier Luna Rosas, Ariadna Rocío Cardoza Rodríguez, Miguel Becerra de Luna, Joaquín Amador Macías.
Instituto Tecnológico de Aguascalientes. Departamento de Sistemas y Computación.
Av. López Mateos 1801 Ote. Fracc. Bona Gens. Aguascalientes, Ags., México.
Teléfono: (01-449) 9105002, Fax: (01-449) 9700423. *fjluna@verona.fi-p.unam.mx*

Resumen

Este artículo propone una nueva forma de balancear la carga dinámicamente combinando un modelo Fuzzy-Logic en conjunto con el algoritmo distribuido Request for Bids (RB), el trabajo es centrado en el modelo Fuzzy-Logic con el propósito de mejorar el balanceo dinámico de carga en un sistema distribuido a gran escala. El modelo Fuzzy-Logic y RB fueron implementados en una arquitectura para balancear la carga bajo el estándar de CORBA y fueron evaluados contra otras estrategias que son bien conocidas en la literatura usando diferentes métricas de carga y mostramos como bajo ciertas condiciones nuestra estrategia obtiene mejores resultados.

Palabras clave

CORBA, Fuzzy-Logic, Dynamic Load Balancing, Request for Bids.

Introducción

El balanceo de carga es parte del amplio problema de asignación de recursos. El problema de balanceo de carga (llamado también planeación de tareas) es cómo distribuir los procesos entre los elementos de procesamiento para lograr algunas metas de desempeño tales como: minimizar el tiempo de ejecución, minimizar los retardos de comunicación, y/o maximizar la utilización de recursos, etc. [5].

Cuando hacemos balanceo de carga aplicamos 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 balanceo de carga en varias formas y en varios niveles del sistema [2]. En la práctica, sin embargo, la efectividad de balanceo de carga depende de la estrategia de distribución de carga escogida, el tipo de métrica de carga y otros parámetros en tiempo de ejecución necesitados por las estrategias.

Las estrategias de balanceo de carga pueden ser divididas en 2 grandes grupos: estrategias de balanceo estático de carga y estrategias de balanceo dinámico de carga. Las estrategias de balanceo estático de carga, obtienen la localización de todos sus procesos antes de comenzar la ejecución. Las estrategias de balanceo dinámico de carga intentan equilibrar la carga en tiempo de ejecución [18].

En el balanceo estático de carga, la asignación de tareas a procesadores es hecha antes de que el programa comience la ejecución. La información respalda los tiempos de ejecución de las tareas y los recursos de procesamiento ya que se asume que son conocidos en tiempo de compilación. Una tarea es siempre ejecutada sobre el procesador al cual fue asignado, podemos decir que los métodos de planeación estáticos son para procesar tareas sin estado de ejecución. La mayor ventaja de los métodos de planeación estáticos, es que toda la sobrecarga (overhead) de la planeación de procesos está incurrida en tiempo de compilación, resultando un tiempo de ejecución más eficiente comparado con los métodos de planeación dinámicos. Sin embargo los métodos de planeación estáticos sufren de muchas desventajas. Tal vez una de las más criticas limitaciones de la planeación estática es que, en general, la planeación óptima genera un problema NP-completo [18].

Por otra parte las estrategias de balanceo dinámico de carga se basan en la redistribución de procesos entre los procesadores durante el tiempo de ejecución [19]. Esta redistribución es ejecutada para transferir tareas de un procesador fuertemente cargado a un procesador ligeramente cargado con el objetivo de mejorar el desempeño de una aplicación. Obtener una solución dinámica es mucho más complicado que obtener una solución estática. Esta requiere agrupamiento y mantenimiento de estado de información en tiempo de ejecución. Sin embargo la asignación dinámica de carga puede lograr un buen desempeño para hacer decisiones de migración de procesos utilizando índices de carga del sistema. Un problema claramente asociado con el agrupamiento y mantenimiento de estado de información es donde las

decisiones serán hechas [8]. La ventaja del balanceo dinámico de carga sobre el balanceo estático de carga, es que el sistema no necesita conocer el tiempo de ejecución de una aplicación antes de su ejecución. La flexibilidad inherente en el balanceo dinámico de carga permite la adaptación de los requerimientos de la aplicación en tiempo de ejecución [18].

Una cantidad significativa de trabajo ha sido hecha sobre servicios de balanceo de carga en varios niveles del sistema, incluyendo la red, el sistema operativo y a nivel middleware [15].

*Balanceo de Carga a Nivel de Red. L*os Servidores de Nombres de Dominio (DNS) y los Ruteadores IP´s que sirven a una gran cantidad de máquinas host proveen este tipo de balanceo de carga [3].

Balanceo de Carga a Nivel del Sistema Operativo. Los Sistemas Operativos Distribuidos generalmente proveen este tipo de balanceo de carga a través del agrupamiento de computadoras, compartición de carga y mecanismos de migración de procesos [5].

Balanceo de Carga Basado en Middleware. Las iteraciones globales son acopladas por arquitecturas llamadas Middleware. La arquitectura más común de Middleware para aplicaciones orientadas a objetos distribuidas es Common Object Request Broker Architecture (CORBA) [12]. 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.

Ha habido importantes enfoques para extender funcionalidades de CORBA que soporten balanceo de carga, por ejemplo, TAO [14], [15], VisiBroker [9], MICO [16], etc. Sin embargo esas iniciativas a menudo han resultado ser diseños para aplicaciones y plataformas específicas.Para consolidar el enfoque previo y resolver este problema la OMG diseño Request For Proposal [13] con el propósito de establecer un estándar de balanceo de carga y monitoreo en aplicaciones basadas en CORBA. El artículo de IONA Technologies [7] pudo ser considerado el trabajo más relevante en esta área y la OMG ha recomendado 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 balanceo de carga.

Específicamente, la estrategia LL fue inicialmente propuesta por IONA [6]; TAO y PrimsTech también ofrecen implementaciones [1]. En nuestra experiencia, LL es un algoritmo de balanceo de carga efectivo y fácil de implementar con un desempeño promedio razonable y de bajo overhead. Un intento por mejorar la estrategia LL es el realizado por Arapé [1], quien trata de ampliar LL para abarcar un sistema de computadoras heterogéneas; aunque no hay resultados reales su propuesta demuestra que es posible realizar este mejoramiento. Luna mejoró la estrategia LL aplicando algoritmos genéticos en [10], la estrategia se implementó en una arquitectura para balancear carga que fue desarrollada bajo el estándar de CORBA y se evaluó con otras estrategias utilizando diferentes métricas de carga; sus resultados demostraron que se puede mejorar LL bajo ciertas condiciones del sistema.

Sin embargo todas estas propuestas se implementaron utilizando balanceo de carga local. De inmediato surge la siguiente pregunta, *¿qué sucede si queremos balancear la carga a gran escala?* Un modelo Fuzzy-Logic en conjunto con el algoritmo distribuido Request for Bids es propuesto para lograr el balanceo de carga a gran escala [20], [21] y [4]. Nuestros resultados simulados preliminares son alentadores ya que demuestran cómo esta propuesta puede ser factible.

El resto de este artículo está organizado como sigue: La teoría de un controlador fuzzy genérico es descrito en la sección 2, el algoritmos distribuido Request for Bids o mejor conocido en inteligencia artificial distribuida con el nombre de contrato net (Contract Net) lo describimos en la sección 3, la combinación del modelo-fuzzy en conjunto con el algoritmo Request for Bids lo podemos ver en la sección 4, los resultados preliminares del modelo-fuzyy los vemos en la sección 5, y finalmente damos nuestras conclusiones.

2. MODELO FUZZY-LOGIC.

La teoría de un controlador fuzzy mostrada en la Figura 1 incluye 5 componentes: fusificación, base de reglas, funciones de membresía, máquina de inferencia fuzzy y defusificación [20], [21].

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 específicas 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.

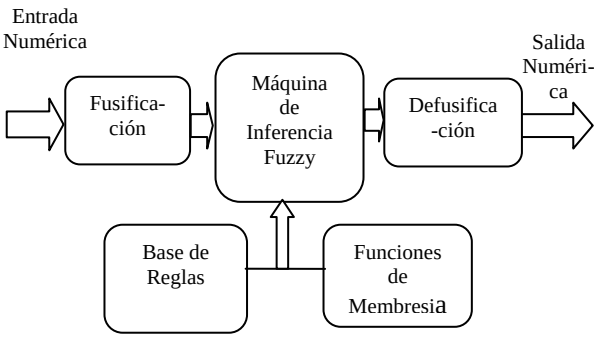

Figura 1. La Arquitectura del Modelo Fuzzy-Logic.

Funciones de Membresía. La función de membresía o pertenencia de un conjunto fuzzy consiste en un conjunto de pares ordenados $F = \{(u, u_F(u)) / u \in U\}$ si la variable es discreta, o una función continua si no lo es. El valor de $u_F(u)$ indica el grado en que el valor u de la variable U está incluida en el concepto representado por la etiqueta F. 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, S, exponencial, π, etc.

Máquina de Inferencia Fuzzy. La Máquina de inferencia fuzzy aplica el mecanismo de inferencia al conjunto de reglas en la base de reglas fuzzy para producir un conjunto fuzzy de salida. Esto envuelve una correspondencia entre el conjunto fuzzy de entrada con las premisas de las reglas, activación de las reglas para deducir la conclusión de cada regla que es disparada, y combinar todas las conclusiones activadas para unirlas en un conjunto fuzzy y después generar el conjunto fuzzy de salida.

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

3. ALGORITMO DISTRIBUIDO REQUEST FOR BIDS.

Generalmente un balanceador de carga debe de ser implementado de manera distribuida para enviar los requerimientos de los clientes a los analizadores de carga que den el servicio más apropiado. El balanceador de carga debe hacer tales decisiones basadas sobre el estado actual de la red y la carga de los demás analizadores.

El modo de asignación de tareas Request for Bids o mejor conocido en inteligencia artificial distribuida con el nombre de contrato net (Contract Net) [4], es el algoritmo que implementa nuestro balanceador para equilibrar la carga en un sistema distribuido a gran escala.

El algoritmo tiene cuatro estados:
1). El primer estado es dedicado a los Request for Bids mismos. El Manager envía una descripción de las tareas a todos los Bidders del sistema.
2). Sobre las bases de la descripción, en un segundo estado, los Bidders obtienen la propuesta y la envían al Manager.
3). El Manager recibe y evalúa las propuestas, y premia el contrato del mejor Bidder en un tercer estado.
4). Finalmente en el cuarto y último estado, el Bidder que ha sido premiado envía un mensaje al Manager para indicarle que aún esta preparado para llevar la tarea requerida y este por lo tanto, se encarga de hacer esto, o si este no puede llevar la tarea a cabo, dispara una re-evaluación de los Bids y se premiará a otro Bidder (y se comienza nuevamente en 3).

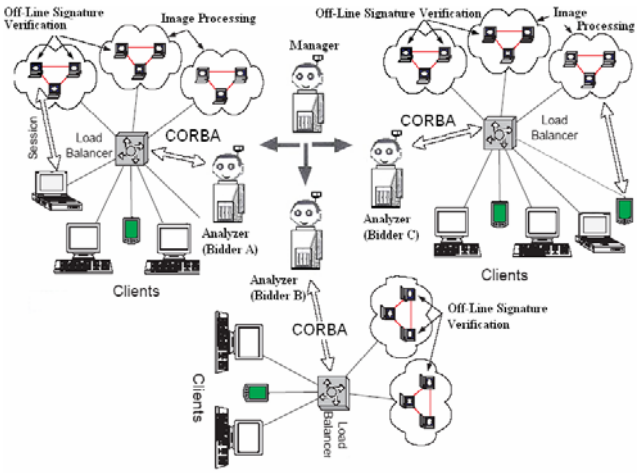

Figura 2 Algoritmo Request for Bids para Balancear Dinámicamente la Carga Entre Varios Analizadores.

4. COMBINANDO EL MODELO FUZZY-LOGIC CON EL ALGORITMO REQUEST FOR BIDS.

La información es intercambiada a través del algoritmo Request for Bids, entre un analizador que se eligió como Manager y los demás analizadores o Bidders (ver Figura 2). Sin embargo la información de estado que guarda el Manager puede no reflejar el estado de los Bidders exactamente debido a los retardos de red [11]. El Manager necesita usar razonamiento aproximado para manejar información fuzzy que haga eficiente el sistema [20], [21].

Para hacer una decisión de balanceo correcta, las variables lingüísticas de la carga de los demás Bidders (LoadBidders), el tiempo de invocación de

métodos remotos (RMIT) y la calidad de servicio (QualityService) es definida en el modelo fuzzy de la siguiente manera:

4.1 CARGA DE LOS BIDDERS (LOADBIDDERS).

Definimos la carga del Bidder, denotada como (LoadBidder), con la definición del conjunto fuzzy: {Low, _espué, High}. Empleamos un enfoque indirecto para medir la carga del Bidder, en vez de medir directamente la carga de cada réplica, medimos la carga del benchmark que es un grupo de réplicas controladas por un analizador de carga. El benchmark esta siempre en ejecución como un proceso background. La Figura 3 muestra la gráfica de membresía para el LoadBidder.

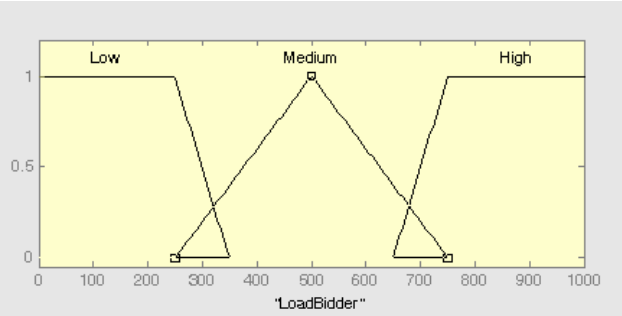

Figure 3 Gráfica de Membresía para la Carga de los Bidders (LoadBidder).

4.2 TIEMPO DEL MÉTODO DE INVOCACIÓN REMOTA (RMIT).

La forma de medir la respuesta de los Bidders al manager así como el overhead introducido, es calculando la utilización de la red. Esto se hace midiendo el tiempo de invocación de métodos remotos (RMI).

Cabe hacer mención que estamos monitoreando el canal de CORBA y dentro del canal de CORBA debemos de considerar los siguientes cuatro puntos en el monitoreo: Send Request, Receive Request, Send Reply and Receive Reply [17].

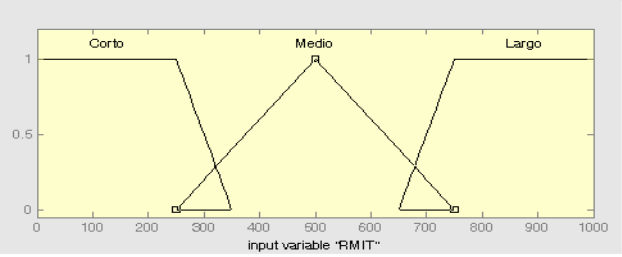

Figure 4 Gráfica de Membresía para el Tiempo de Respuesta Remoto (RMIT).

Estos cuatro puntos están inmersos en la llamada al método RMI y definimos este método simplemente retornando un tipo de datos primitivo del bidder al manager midiendo el tiempo. El método System.currentTime() es usado para medir el tiempo de enlace durante la invocación del método remoto, y está calculado en milliseconds. El conjunto fuzzy para Remote Method Invocation Time (RMIT) esta definido como: {Short, _espué, Long}. La Figura 4 muestra la gráfica de membresía para medir el tiempo de invocación del método remoto (RMIT).

5.3.3 CALIDAD DEL SERVICIO (QUALITYSERVICE).

Usamos QualityService para clasificar servicios en seis diferentes categorías: {VeryLow, Low, MediumLow, Medium, MediumHigh, High}. La gráfica de membresía es mostrada en la Figura 5.

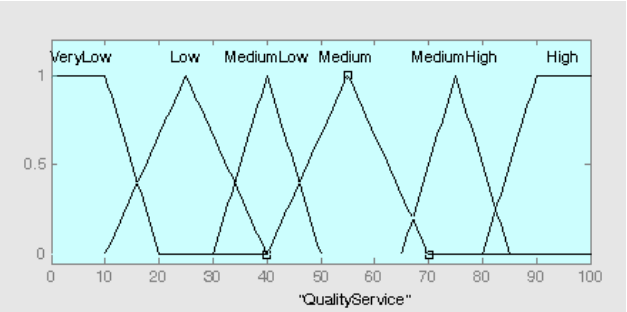

Figure 5 Gráfica de Membresía para la Calidad del Servicio (QualityService).

_espués de definir las variables fuzzy un conjunto de reglas de inferencia es definido como sigue:

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

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 LoadBidder es Low, entonces QualityService es High (ver Figura 6). 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 LoadBidder 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 llevado a cabo. Usamos el método del centroide para traslapar las áreas [20], [21].

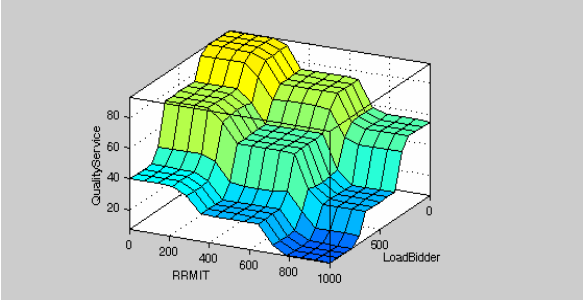

Figure 6 Grafica de Salida para QualityService.

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 al mejor Bidder basándose en la QualityService obtenida para cada Bidder. En la Figura 6 observamos la QualityService si tenemos diferentes entradas que representan la carga en los Bidders y el tiempo de respuesta que genera cada uno en la red.

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

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 los Bidders (LoadBidder), Tiempo de Respuesta en la Red (RMIT) y Calidad del Servicio (QualityService).

En la entrada uno observamos claramente que cuando nuestros Bidders manejan una carga elevada y el tiempo de respuesta de la red se encuentra en un tiempo medio, nuestro sistema infiere que QualityService no es muy buena (25%).

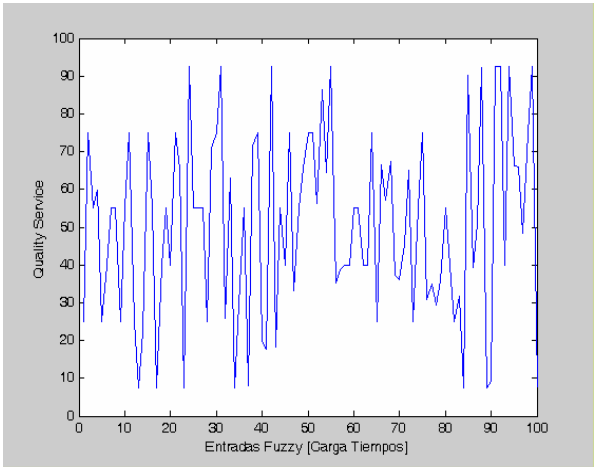

Figura 7 Quality of Service de los Bidders.

En la segunda entrada nuestros Bidders tienen baja carga, pero siguen manteniendo la respuesta de la red en un tiempo medio, por lo tanto nuestro sistema infiere que la QualityService es medio alta (75%). En la entrada 13 y 17 cuando los Bidders tienen un tiempo de respuesta alto y una carga elevada, nuestro sistema

infiere que la QualityService es muy mala (8%). La gráfica 7 muestra la QualityService que genera nuestro modelo fuzzy cuando simulamos 100 entradas de los Bidders.

Conclusiones

Cuando existe más de una LAN el balanceo de carga debe ser escalado de tal forma que podamos lograr el equilibrio de carga en todo el sistema distribuido. Para lograr esto utilizamos un modelo de equilibrio de carga basado en Fuzzy-Logic en conjunto con el algoritmo distribuido Request for Bids. Este artículo describe la arquitectura del modelo Fuzzy-Logic, los cuatro pasos fundamentales del algoritmo distribuido Request for Bids y por último la manera en que se combinan, para lograr el equilibrio de carga a gran escala, simulamos la carga de varios bidders y el tiempo de respuesta que genera cada uno en la red, que sirven como antecedentes para aplicar las reglas de inferencia y las gráficas de membresía en nuestro modelo fuzzy, esos valores son comparados y el mínimo de los dos es entonces proyectado sobre la función de membresía de su gráfica consecuente. El proceso de defusificación generó un valor centroide que representa el rango de la QualityService. El Manager puede entonces enviar los requerimientos del cliente al mejor Bidder basándose en la Quality Service obtenida para cada bidder.

Referencias

[1] Nelson Arapé, Juan Andrés Colmenares, and Nestor V. Queipo. On the Development of an Enhanced Least Loaded Strategy for the CORBA Load Balancing and Monitoring Service. In Proc. 16th Int'l Conference on Parallel and Distributed Computing Systems. Reno, Nevada, USA. August, 2003.

[2] J. Balasubramanian, D. C. Schmidt, D. Lawrence, O. Ossama, Evaluating the Performance of Middleware Load Balancing Strategies, in: Proc. 2004, Enterprise Distributed Object Computing Conference, Eighth IEEE International (EDOC'04), 2004, pp. 135-146.

[3] Cisco Systems, Inc., High Availability Web Services, www.cisco.com/warp/public/cc/so/neso/ibso/s390/mnibm_wp.htm, 2000.

[4] Ferber Jacques. Multi-Agent Systems. An Introduction to Distributed Artificial Intelligence, Addison-Wesley, 1999.

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

[6] IONA Technologies, Tri-Pacific Software Inc, VERTEL Corporation, Load Balancing and Monitoring. Initial Joint Submission (orbos/01-08-05), Aug. 2001.

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

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

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

[10] Luna Rosas Fco. Javier, Alcantará Silva Rogelio. 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, June 27-30, 2005.

[11] Mirchandaney R., Towsley D., Stankovic J. A., Analysis of the effects of delays on load sharing, IEEE Trans. Comput. 38(11) November 1989, 1513-1525.

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

[13] Object Management Group, Inc. Load Balancing and Monitoring for CORBA-based Applications. Request for Proposal (orbos/2001-04-27), Apr. 2001

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

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

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

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

[18] B. A. Shirazi, A. R. Hurson, and M. K. Kavi. Sheduling and Load balancing in Parallel and Distributed Systems. IEEE Computers Society Press, Los Alamitos, California USA, 1995.

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

[20] Yen John, Langari Reza. FUZZY LOGIC, Intelligence, Control and Information. Prentice Hall, 1999.

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