Conciencia Tecnológica No. 36, Julio - Diciembre 2008

Arquitectura para Balancear Carga Dinámica por Demanda Adaptativa, utilizando CORBA en JAVA-IDL

Investigación

Ing. Jesús Chávez Esparza[1], M. en C. Gerardo Rentería Castillo[2], Dr. Francisco Javier Luna Rosas[2]
[1]Pasante de Maestría en Ciencias en Ciencias Computacionales por el Instituto Tecnológico de Toluca, Av. Instituto Tecnológico s/n, Ex-Rancho La Virgen, Metepec, Estado de México, México;
e-mail: j_chavez2000@hotmail.com
[2] Instituto Tecnológico de Aguascalientes, Av. Adolfo López Mateos 1801 Ote. Esq. Av. Tecnológico, Fracc. Bonagens, C.P. 20256, Aguascalientes, Ags., México;
e-mail: renterg@yahoo.com, fjluna@ita.mx

Resumen

En este trabajo desglosaremos la elaboración de una nueva Arquitectura para balancear carga dinámica, por demanda adaptativa, utilizando CORBA en JAVA-IDL.

Una arquitectura de balanceo 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.

A través de las pruebas se justifica el uso de la arquitectura y se definen los parámetros a considerar para obtener un óptimo desempeño, refiriéndose al número de réplicas, al tipo de requerimiento y la estrategia de balanceo.

Palabras claves: CORBA, Balanceo de carga.

Abstract

In this work breakdowns developing a new architecture for dynamic load balancing, adaptive demand by using a Java-Corba IDL.

A load balancing architecture is a system that allows you to distribute the computational work among multiple computers, with the aim of reducing the response time overall system.

Through the evidence justifies the use of architecture and defines the parameters to be considered for an optimal performance, referring to the number of replicas, the type of requirements and the strategy of balancing.

Keywords: CORBA, Load balancing

1. 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, con la finalidad de procesar un requerimiento y recibir una respuesta adecuada [1]. Los servidores están en ejecución continuamente, y los clientes lo hacen solamente cuando están realizando invocaciones remotas.

Los sistemas distribuidos tienen las siguientes características:

- Concurrencia de los componentes.
- Carencia de reloj global.
- Fallas independientes en sus componentes.

Los sistemas distribuidos tienen como objetivo superar los siguientes desafíos:

- Heterogeneidad. Permite la variedad y diferencia aplicables a las redes, hardware de computadoras, sistemas operativos y lenguajes de programación.
- Extensibilidad. Determina por el grado de permitir añadir nuevos servicios para compartir recursos y ponerlos a la disposición del cliente.
- Seguridad. La seguridad en recursos de información tiene tres componentes: confiabilidad, integridad y disponibilidad.
- Escalabilidad. El uso de datos replicados e implementación de múltiples servidores, permiten incrementar los recursos disponibles en el sistema.
- Tratamiento de fallas. Los sistemas computacionales a veces fallan y pueden producir resultados incorrectos.
- Concurrencia. Tanto los servicios como las aplicaciones proporcionan recursos que pueden compartirse entre los clientes de un sistema distribuido.
- Transparencia. Se define la transparencia como la ocultación al usuario o programador de aplicaciones de la separación de los componentes en un sistema distribuido.

Nuestra arquitectura debe satisfacer la concurrencia y fallas independientes de sus componentes. Además nuestra arquitectura será elaborada en CORBA para cumplir los desafíos de un sistema distribuido.

2. Arquitectura común del intermediario para la petición de objetos CORBA

CORBA es una arquitectura que permite la integración de una gran variedad de sistemas de objetos. CORBA es una norma, no un producto [1]. CORBA se encarga de especificar, en un entorno distribuido heterogéneo, el intercambio de operaciones entre objetos de manera transparente. El encargado de transportar las

llamadas de los requerimientos del cliente y traducirlas para su ejecución es el ORB.
El ORB es el encargado de comunicar los requerimientos de los clientes, en el envío y retorno de respuestas. Los objetos pueden estar corriendo en una máquina, dentro de la red o sobre la misma máquina.

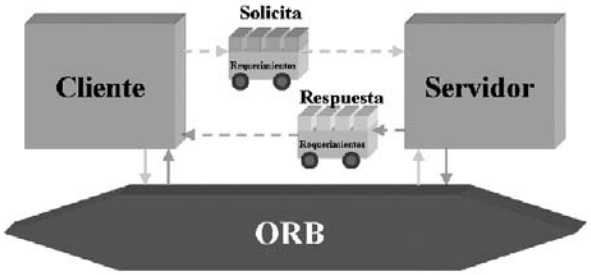


Figura 1. Comportamiento de los objetos CORBA.

CORBA utiliza el lenguaje de definición de interfaces (IDL) para proporcionar los tipos de operaciones, sus parámetros y generando la estructura de las interfaces de la implementación. Debido a que CORBA es independiente del lenguaje de programación esto permite la posibilidad de integrar diferentes lenguajes de programación en una sola aplicación. El lenguaje de definición de interfaces (IDL), al interpretarlo, permite a cada lenguaje de programación (para nuestro caso JAVA) obtener la especificación correcta para cada operación.

CORBA permite la comunicación entre los elementos de la arquitectura transparentemente, pero la elaboración de una arquitectura de balanceo de carga, requiere contar con un algoritmo de balanceo para equilibrar la carga de trabajo entre los servidores.

3. Algoritmos de balanceo de carga

Singhal [5] clasifica los diferentes algoritmos de balanceo de carga en: estáticos, dinámicos y adaptativos. Los algoritmos de balanceo de carga dinámicos usan la información del estado del sistema, por ejemplo la longitud de cola del servidor, para hacer decisiones de distribución de carga, mientras los algoritmos de balanceo de carga estáticos no usan esta información. Los algoritmos de balanceo de carga dinámicos generan más sobrecarga por recolectar, almacenar y analizar la información del estado del sistema, pero tienen mejor potencial de balanceo que los algoritmos estáticos. Los algoritmos de balanceo de carga adaptativos, son algoritmos dinámicos, con la diferencia de que cambia sus parámetros adaptando sus actividades, dependiendo del estado del sistema. Ello incluye el poder suspender la colección de información cuando la carga global sea alta, para evitar sobrecargar el sistema.

Son deseables en los algoritmos de balanceo las siguientes propiedades:

- No generar una sobre carga computacional por el mismo algoritmo.
- Que el algoritmo tenga un comportamiento estable, es decir que con carga ligera y pesada sea óptimo.
- La principal, generar el buen desempeño del sistema con mejores tiempos y óptimo uso de recursos.

El balanceo de carga es la parte de la política de planeación distribuida y tiene como finalidad el mejorar el desempeño y eficiencia del sistema.

4. Arquitecturas de balanceo de carga con CORBA

Las arquitecturas 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.

Las arquitecturas de balanceo de carga con CORBA se componen de los siguientes elementos básicos:

- Clientes.
- Réplicas.
- Balanceador de carga.

Hay varias estrategias para diseñar en CORBA servicios de balanceo de carga [4] y pueden ser clasificadas de la siguiente manera:

- Por sesión.
- Por requerimiento.
- Sobre demanda.

Cuando se diseña un servicio de balanceo de carga es importante seleccionar un algoritmo adecuado que decida la réplica en donde se procesará el requerimiento que llega. Esto hace necesario generar políticas de balanceo de carga, las cuales de forma general pueden ser clasificadas [4] dentro de las siguientes:

- No adaptativa.
- Adaptativa.

Combinando las estrategias y las políticas de balanceo de carga descritas anteriormente es posible crear seis diferentes arquitecturas de balanceo de carga:

1. Arquitectura de balanceo de carga por sesión no adaptativa.
2. Arquitectura de balanceo de carga por requerimiento no adaptativa.
3. Arquitectura de balanceo de carga por demanda no adaptativa.
4. Arquitectura de balanceo de carga por sesión adaptativa.
5. Arquitectura de balanceo de carga por requerimiento adaptativa.
6. Arquitectura de balanceo de carga por demanda adaptativa.

De las seis arquitecturas diferentes, consideramos como la mejor a la arquitectura de balanceo de carga por demanda adaptativa, porque al utilizar demandas se reduce la operación en el balanceador y al adaptar sus parámetros del sistema se evita saturar las réplicas, además ofrece una clara ventaja ya que puede responder a los cambios dinámicos en la carga del cliente, y se tiene un análisis de la efectividad de la arquitectura llevado a cabo por Ossama [6].

5. Análisis y diseño de la arquitectura de balanceo de carga

Los componentes del balanceador de carga funcional son descritos a continuación para proveer una mejor idea de cómo la implementación del balanceador de carga bajo CORBA debería ser vista para operar [3].

- Localizador de réplicas
- Monitor de carga
- Analizador de carga
- Balanceador de carga

Para el análisis y diseño de nuestra propuesta, la captura de requisitos de arquitectura de balanceo de carga la realizaremos utilizando el diagrama de casos de uso de UML. Analicemos los requerimientos del cliente (Figura 2). El cliente físico del sistema precisa conocer los requerimientos que puede resolver la arquitectura. Además, el cliente necesita poder solicitar esos requerimientos a la arquitectura y recibir el resultado esperado de ellos.

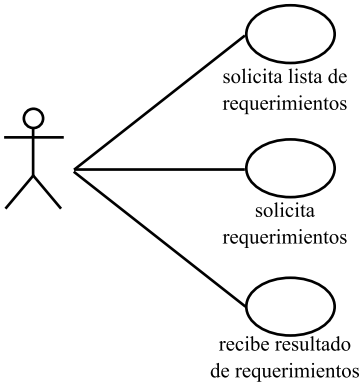


Figura 2. Diagrama de casos de uso.

El Diagrama de Clases es el diagrama principal para el análisis y diseño del sistema. Un diagrama de clases representa las clases del sistema con sus relaciones estructurales y de herencia. La definición de clase incluye las definiciones de atributos y operaciones. El modelo de casos de uso aporta información para establecer las clases.

El diagrama de clases de nuestra arquitectura de balanceo de carga está representado en la Figura 3 con un grado abstracción.

En una arquitectura de balanceo de carga pueden existir varias réplicas y clientes, pero se recomienda utilizar solo un balanceador.

Los diagramas de secuencias son modelos que describen la manera en que colaboran grupos de objetos para cierto comportamiento [2]. El diagrama de la Figura 4 muestra el comportamiento de la arquitectura de balanceo de carga en la secuencia de las operaciones, a un alto nivel de abstracción.

En el diagrama de la Figura 5 se observa la asignación de réplicas a los diferentes clientes.

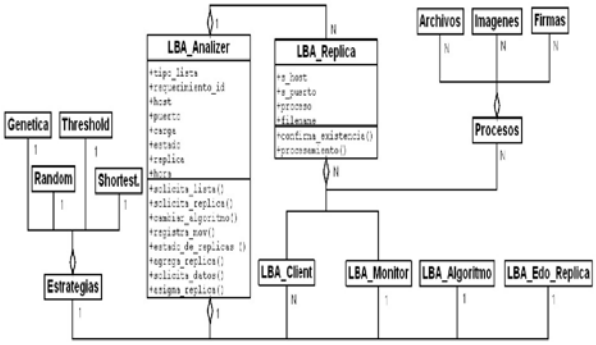


Figura 3. Diagrama de clases de la arquitectura.

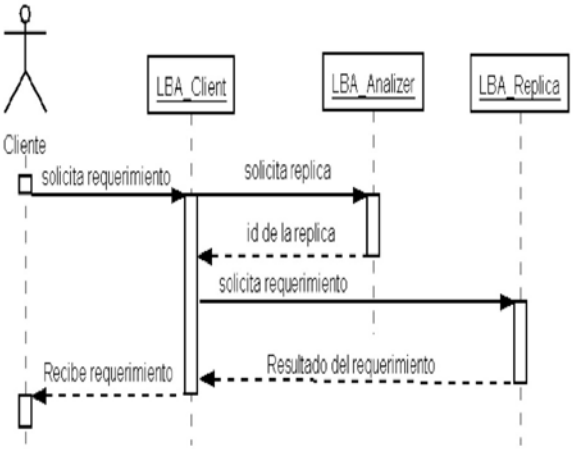


Figura 4. Diagrama de secuencia del cliente.

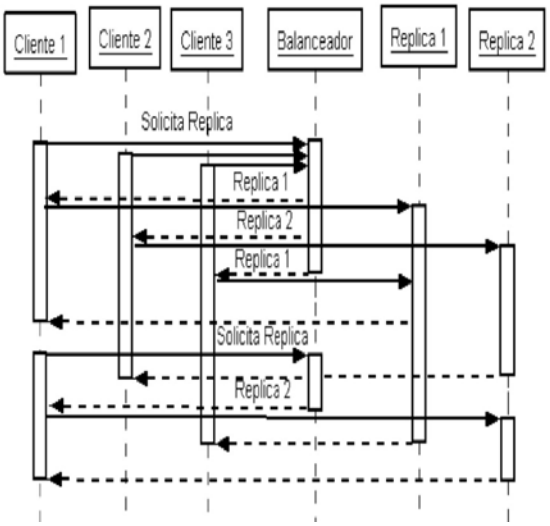


Figura 5. Diagrama para selección de réplicas.

En el diagrama de colaboración de la Figura 6 podemos observar que el LBA_Analizer concentra la mayor cantidad de métodos para administrar la arquitectura y que las réplicas tienen como operación principal satisfacer los requerimientos de los clientes.

Esto tiene su origen en el objetivo de una buena arquitectura de balanceo de carga de satisfacer los requerimientos de los clientes a la mayor eficiencia posible.

En el diagrama de actividades (Figura 7) se puede observar la implementación de la arquitectura al inicio de una sesión.

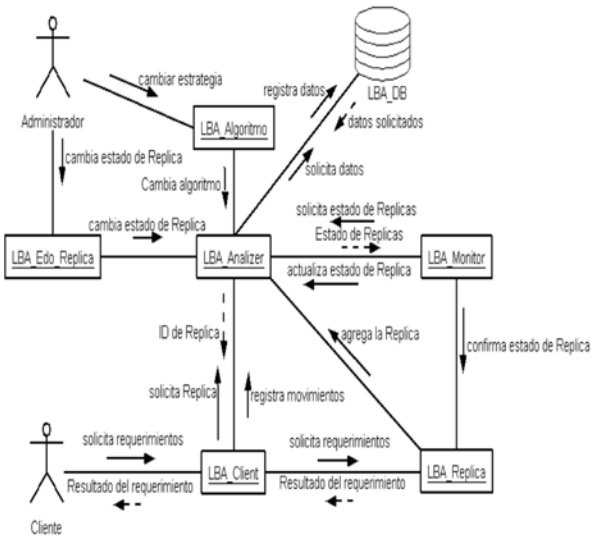


Figura 6. Diagrama de colaboración.

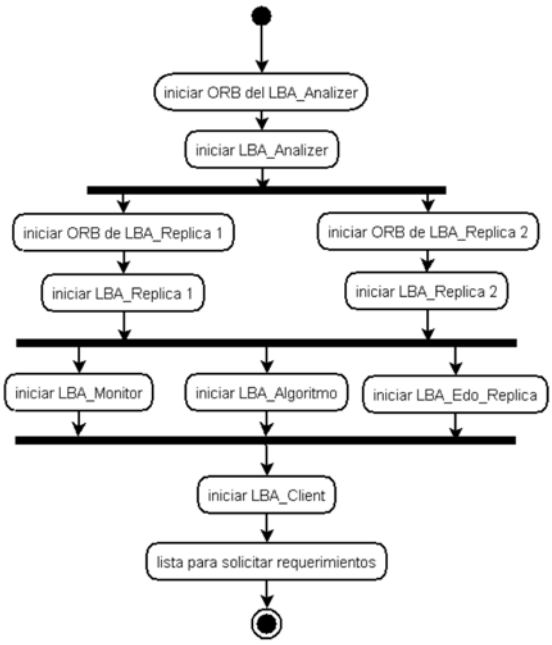


Figura 7. Diagrama de actividades.

Nuestra arquitectura permite balancear requerimientos diferentes, utilizando varias estrategias de balanceo que se pueden cambiar en tiempo de ejecución. Además nuestra arquitectura permite incrementar o disminuir el número de réplicas en tiempo de ejecución con la finalidad de mejorar la eficiencia.

Nuestra arquitectura permite al usuario solicitar un requerimiento que le será surtido en el menor tiempo posible gracias al balanceo de la carga, sin importarle en dónde se encuentra ubicada la réplica que surte ese requerimiento, ni cuántos requerimientos se están surtiendo al mismo tiempo. El usuario, gracias a las propiedades de CORBA, recibe el resultado del requerimiento como si estuviera corriendo un proceso local.

6. Pruebas y conclusiones

La eficacia de la arquitectura de balanceo de carga la demostraremos respondiendo a las siguientes preguntas:

¿Porqué usar una arquitectura de balanceo de carga?

Se ha realizando la prueba de enviar 2400 requerimientos, primeramente a una réplica tomando el tiempo total que se emplea para solucionar la demanda y después se fueron incrementando el número de réplicas, hasta determinar la gráfica de la figura 8.

Figura 8. Prueba de incremento de réplicas.

Como podemos observar en la figura 8 al ir incrementando el número de réplicas el tiempo total disminuye drásticamente por lo que podemos concluir que una arquitectura de balanceo de carga que utiliza varias réplicas permite disminuir el tiempo global del sistema.

¿Cuántas réplicas debe usar una arquitectura?

Partiendo de la prueba anteriormente mencionada cuando se incrementan el número de réplicas el tiempo total disminuye (ver figura 9). Por la gráfica podemos concluir que el agregar réplicas disminuye el tiempo total de la arquitectura, sin embargo, llega un momento en que agregar más réplicas no disminuye el tiempo total.

Por lo tanto podemos concluir que la mejor opción para esta aplicación es cuatro réplicas y tener más de ocho no sería muy apropiado. Pero, el administrador de la arquitectura, debe realizar el análisis para cada aplicación y determinar el número óptimo de réplicas.

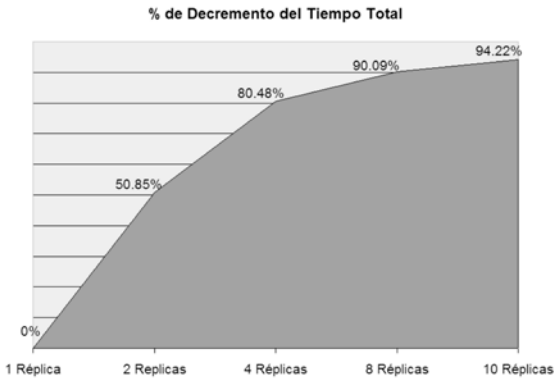
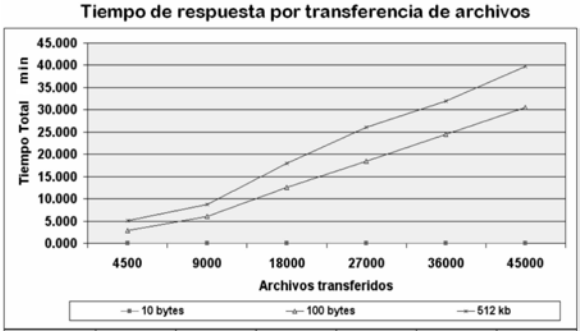


Figura 9. Porcentaje de disminución del tiempo.

¿La arquitectura funciona igual con cualquier requerimiento?

Para contestar esta pregunta se realizó una prueba en donde se transfería: 4500, 9000, 18000, 27000, 36000 y 45000 archivos de 10 Bytes (B), 100 B. y 512 KB. En la figura 10 se muestran los resultados de la prueba.



Archivos	4500	9000	18000	27000	36000	45000
10 bytes	0.046	0.039	0.045	0.045	0.049	0.048
100 bytes	2.784	6.050	12.615	18.463	24.410	30.484
512 kb	5.025	8.682	18.029	26.045	32.020	39.690

Figura 10. Tiempo de procesamiento por archivos transferidos.

De los datos podemos observar que **el tiempo total no es proporcional al tamaño del archivo**. Con respecto al número de archivos transferidos si muestra un comportamiento de proporcionalidad.

Esto demuestra que el comportamiento de la arquitectura es diferente dependiendo del requerimiento que esté procesando. En el caso de transferencia de archivos influye otro factor importante que es el ancho de banda de la red. Cuando la transferencia de archivos sea mínima el ancho de banda no es importante.

¿Cuáles estrategias de balanceo puede usar la arquitectura?

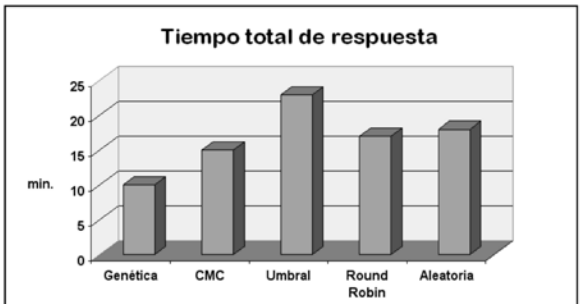
La arquitectura de balanceo de carga fue probada con varias estrategias de balanceo de carga entre las que destacan la estrategia Genética, Cola más corta (CMC), Umbral, Round Robin y Aleatoria.

Sin embargo, se puede utilizar cualquier estrategia que permita equilibrar los datos de las demandas entre las réplicas disponibles.

¿La arquitectura funciona igual con todas las estrategias?

Para contestar esta pregunta se probó la arquitectura de balanceo de carga con el operador Laplace a 800 imágenes heterogéneas en diferentes formatos (GIF y JPEG), como podemos observar, en la figura 11, existe variación en el tiempo de respuesta siendo el menor con la estrategia genética.

Esto demuestra que la arquitectura responde con diferente tiempo total en cada estrategia de balanceo de carga.



Estrategia	Genética	CMC	Umbral	Round Robin	Aleatoria
Tiempo (min.)	10.02	15.02	23.03	17.03	18.03

Figura 11. Tiempo de procesamiento distribuido de imágenes (Operador Laplace).

Conclusión global

Decidimos elaborar una nueva arquitectura de balanceo de carga con el objetivo de reducir el tiempo global del sistema, y se ha comprobado que al utilizar la arquitectura con varias réplicas el tiempo global disminuye drásticamente lo que confirma nuestra hipótesis.

El uso de la arquitectura depende básicamente del requerimiento que se desea implementar y ello lleva a realizar previamente pruebas con diferentes algoritmos y número de réplicas para seleccionar el óptimo.

Referencias

[1] Coulouris George, Dollimore Jean, Kindberg Tim; (2001); *Sistemas Distribuidos conceptos y diseños* 3ª Edición; Addison Wesley.
[2] Fowler Martin, Kendall Scott; (1997); UML, Gota a Gota; Addison Wesley.
[3] IONA Technologies, (2002); Tri-Pacific Software

Inc, VERTEL Corporation, Load Balancing and Monitoring, Supported by Alcatel, Institute National of 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.

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

[5] Singhal Muskesh and Shivaratri G. Niranjan (1994); *Advanced Concepts in Operating Systems (Distributed, Database and Multiprocessor Operating Systems)*. McGraw-Hill.

[6] Ossama Othman, Carlos O'Ryan, and Douglas C. Schmidt; *An Efficient Adaptive Load Balancing Service for CORBA;* Department of Electrical and Computer Engineering; University of California, Irvine; February 13, 2001.

Artículo recibido: 10 de junio de 2008
Aceptado para publicación: 3 de noviembre de 2008