UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

PROGRAMA DE MAESTRIA Y DOCTORADO EN INGENIERIA

FACULTAD DE INGENIERIA

Balanceo Dinámico de Carga en Ambientes Distribuidos Bajo CORBA

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

DOCTOR EN INGENIERIA

CAMPO DE CONOCIMIENTO – ELECTRICA

P R E S E N T A:

FCO. JAVIER LUNA ROSAS

TUTOR:

ROGELIO ALCANTARA SILVA

2006

Dedicatoria.

Esta Tesis es Dedicada a Gris y Marisol por su Amor y Soporte.

JURADO ASIGNADO:

Presidente:	**DR. FRANCISCO J. GARCIA UGALDE**
Secretario:	**DR. JESUS SAVAGE CARMONA**
1er. Vocal	**DR. ROGELIO ALCANTARA SILVA**
2do. Vocal	**DR. BORIS ESCALANTE RAMIREZ**
3er. Vocal	**DRA. GRACIELA ROMAN ALONSO**
1er. Suplente	**DR. JESUS A. SANCHEZ VELAZQUEZ**
2do. Suplente	**DR. FELIX F. RAMOS CORCHADO**

Lugar o lugares donde se realizó la tesis:

DIVISIÓN DE ESTUDIOS DE POSGRADO FACULTAD DE INGENIERIA (DEPFI-UNAM)

TUTOR DE TESIS:
NOMBRE

_____

FIRMA

Índice

Capítulo 3
Algoritmos para Balancear Carga Dinámicos

Capítulo 4
Estrategias y Arquitecturas para Balancear Carga en CORBA

Capítulo 5
Diseño de la Arquitectura para el Balanceo Dinámico de Carga Bajo CORBA

Capítulo 6
Análisis Comparativo de Estrategias de Balanceo Dinámico de Carga Bajo CORBA

Capítulo 7
Análisis Comparativo de Estrategias en el Preprocesamiento Distribuido de Imágenes

Capítulo 8
Análisis Comparativo de las Estrategias de Balanceo de Carga en un sistema de Verificación Automática de Firmas Manuscritas Off-line.

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.

En el balanceo estático de carga, la asignación de tareas a procesadores es hecha antes de que el programa comience la ejecución. La información (tiempos de ejecución y retardos de comunicación) de las tareas se asume que es conocida 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 [Lo 88, Sarkar 86, Shirazi 90 y Stone 77] en Behrooz A. Shirazi [Shirazi 95].

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.

Por otra parte los algoritmos de balanceo dinámico de carga se basan en la redistribución de procesos entre los procesadores durante el tiempo de ejecución. 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. La solución dinámica 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 tomar 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, dónde las decisiones serán tomadas?.

La ventaja de la distribución dinámica de carga sobre la distribución estática de carga, es que el sistema no necesita conocer el tiempo de ejecución de una aplicación antes de su ejecución. La flexibilidad inherente de la distribución dinámica de carga permite la adaptación de los requerimientos de la aplicación en tiempo de ejecución. La distribución dinámica de carga es particularmente usada en un sistema consistente de una red de estaciones de trabajo en el cual la meta es maximizar la utilización del procesador minimizando el tiempo de ejecución de las aplicaciones.

Las desventajas de los esquemas de balanceo dinámico de carga es la sobrecarga (overhead) en tiempo de ejecución debido a:

1. La información de carga transferida entre los procesadores.
2. Los procesos que toman decisiones para la selección de procesos y procesadores

para transferir trabajos.
3. Los retardos de comunicación generados por la relocalización de tareas.

Este trabajo como ya mencionamos previamente propone una nueva estrategia para balancear dinámicamente carga en CORBA aplicando algoritmos genéticos. El reporte consta de ocho capítulos y dos anexos que describen lo siguiente:

El capítulo I tiene como principal objetivo describir de una forma introductoria los principales componentes de CORBA, objetivo y función tal como lo describe [Orfali 98, Vogel 98, López 01 y OMG 02]. CORBA es el resultado de la solicitud de un consorcio llamado Object Management Group (OMG) que es una organización internacional apoyada por más de 600 miembros, incluyendo vendedores de sistemas de información, diseñadores de software y usuarios. Fundada en 1989, la OMG promueve la teoría y práctica de la tecnología orientada a objetos en el desarrollo de software. CORBA es la especificación que permite integrar una gran variedad de sistemas de objetos, con CORBA el cliente envía el requerimiento a la aplicación de objetos y ésta se encarga de realizar la operación. La importancia de CORBA radica en que permite definir la conexión prácticamente de cualquier forma existente cliente / servidor.

En el capítulo II se ilustra una taxonomía de los algoritmos de balanceo de carga tal como lo describe Casavant y Kuhl [Casavant 88a], la taxonomía describe varias clases de algoritmos en base a un esquema de clasificación jerárquica, los algoritmos que no pueden ser clasificados jerárquicamente son clasificados en forma plana (flat). El capitulo resalta la diferencia que existe entre las políticas que mantienen su estado de ejecución con las políticas que no mantienen su estado de ejecución de acuerdo con Hisao Kameda en [Kameda 97]. El capitulo concluye mostrando varias metas que se siguen en la distribución de carga tal como lo mencionan Mukesh Singhal y Niranjan G. Shivaratri [Singhal 94], A. Goscinski [Goscinski 94], Hisao Kameda [Kameda 97] y Behrooz A.Shirazi [Shirazi 95].

El capítulo III tiene como objetivo ilustrar diferentes algoritmos de balanceo dinámico de carga que han aparecido en la literatura y que caen en la clasificación taxonómica mencionada en el capítulo II. Andrzej Goscinski selecciona cinco algoritmos que corresponden a la clasificación jerárquica [Goscinski 92]. Eager, Lazowska y Zohorjan cubren tres algoritmos simples pero aún efectivos que caen en la clasificación plana. Estos algoritmos son emisores, en esta clase de algoritmos la actividad de distribución de carga es iniciada por un nodo sobrecargado (emisor) que intenta enviar una tarea a un nodo descargado (receptor). Los algoritmos son el Aleatorio, Cola Más Corta y Umbral [Eager 86]. El capítulo concluye mencionando algunos algoritmos y políticas que cubren un enfoque adaptativo y simétrico propuestos por Krueger [Krueger 94].

El capítulo IV muestra las arquitecturas de balanceo de carga que podemos diseñar en CORBA combinado tres estrategias de balanceo de carga (Por-sesión, Por-requerimiento y Sobre-demanda) con dos políticas (Adaptativa y No-adaptativa). Tal como lo menciona Ossama y Douglas C. Schmidt en [Ossama 01a, 01b, 01c, 03a , 03b, IONA 02 y Arápe 03]. El capítulo concluye mencionando los elementos estructurales que debe tener todo sistema de balanceo de carga diseñado en CORBA con el propósito de tener una mejor idea de cómo la implementación de balanceadores de carga debería de ser vista para operar [IONA 02].

El capítulo V muestra el análisis, diseño y construcción de la arquitectura que permiten el balanceo dinámico de la carga bajo CORBA. La arquitectura se apoya en el lenguaje de modelado unificado (UML) [Booch 94, 99, Fowler 99], en el estándar de objetos distribuidos CORBA [OMG 99, 01, 02] y en el lenguaje de definición de interfaces JAVA-IDL, que permite la integración del lenguaje de programación JAVA en aplicaciones distribuidas CORBA [OMG 99, Vogel 98].

En el capítulo VI mostramos los resultados que arrojó un comparativo de cuatro estrategias de balanceo dinámico de carga versus una estrategia Genética, las estrategias se implementaron bajo CORBA. Implementamos las tres estrategias propuestas por [IONA 02]: Round Robin, Random (Aleatoria) y Cola Más Corta "Least_Loaded" (CMC). Adaptamos una estrategia Umbral de los algoritmos emisores según Eager, Lazowska y Zohorjan [Eager 86] y por último diseñamos nuestra estrategia Genética. Una aplicación fue implementada cuyo objetivo fue la transferencia de archivos, 4500, 9000, 18000, 27000 y 36000 archivos fueron evaluados con diferente granularidad (peso) y observamos que a medida que aumentamos la granularidad nuestra estrategia obtuvo mejores resultados, partiendo de esta observación el balanceo dinámico de carga heterogéneo solamente se evaluó con archivos de granularidad gruesa, una aplicación fue implementada cuyo objetivo fue la transferencia de archivos, 900 archivos con granularidad gruesa fueron evaluados, cabe mencionar que no consideremos la misma cantidad de archivos por restricciones de almacenamiento. El comparativo se evaluó utilizando diferentes métricas de carga y se ilustro cómo la estrategia genética bajo ciertas condiciones del sistema arrojó mejores resultados. Este trabajo se fundamentó principalmente en Othman Ossama and Douglas C. Schmidt en [Ossama 01a, 01b, 01c, 03a, 03b], IONA Technologies [IONA 02], Nelson Arapé, Juan Andrés Colmenares, and Nestor V. Queipo. [Arapé 03], Hisao Kameda [Kameda 97], Grady Booch [Booch 94, 99], Behrooz A. Shirazi [Shirazi 95], y Randy L. Haupt [Haupt 98].

Una vez detectadas las mejores condiciones del sistema implementamos dos aplicaciones con requerimientos de granularidad gruesa: *a) una aplicación para el procesamiento distribuido de imágenes, b) una aplicación para la verificación automática de firmas manuscritas estáticas (off-line).*

En el capítulo VII se realiza un análisis comparativo de las estrategias de balanceo de carga en el procesamiento distribuido de imágenes. Evaluamos las estrategias aplicando diversos operadores individuales (negativo, escala de grises, la potencia para abrillantar u oscurecer una imagen y la binarización de la imagen) [Lyon 99, González 95 y Pajares 04] para transformar el valor de cada píxel en la imagen de entrada y generar una nueva imagen de salida con los píxeles transformados. Evaluamos las estrategias implementado el algoritmo de Zhang y Suen [Lyon 99] con el propósito de adelgazar (esqueletizar) los contornos de una imagen binaria. Por último implementamos operaciones basadas en múltiples puntos (vecindad de píxeles) con el propósito de realizar la detección de bordes basada en la convolución [Lyon 99 y González 95]. Evaluamos las estrategias implementado el operador Prewitt, Laplaceano y Roberts con el propósito de detectar bordes para delinear los límites de los objetos en la imagen.

Finalmente el capítulo VIII tiene como objetivo mostrar un análisis comparativo de las estrategias de balanceo de carga en un sistema de verificación de firmas *off-line* [Bigus 01], [Hilera 00], [Justino 01], [Lee 96], [Martínez 04] y [Plamondon 00]. El sistema de verificación fue basado sobre una arquitectura que opera en dos fases, la fase de

entrenamiento y la fase de verificación. La fase de entrenamiento es formada por los estados de extracción de la firma, preprocesamiento y extracción de características y la generación del modelo. La fase de verificación tiene etapas que son altamente similares a la fase de entrenamiento, en esta se comparten las tres primeras etapas y en la cuarta etapa las características extraídas de la firma son comparadas utilizando un clasificador de redes neuronales artificiales BPNN (Back-Propagation Neural Network) versus las características obtenidas en la fase de entrenamiento. Después de conocer el análisis comparativo de nuestras estrategias en el sistema de verificación, mostramos el trabajo a futuro que pretendemos realizar para lograr el equilibrio de carga en un sistema distribuido de gran escala. Un modelo Fuzzy-Logic en conjunto con el algoritmo distribuido Request for Bids es propuesto por nosotros para lograr dicho equilibrio de carga [Yen 99], [Yu 04] y [Ferber 99]. Los resultados simulados preliminares son alentadores ya que demuestran como nuestra propuesta puede ser factible, concluimos el capitulo dando a conocer nuestras conclusiones generales y algunas publicaciones que se realizaron acerca de nuestro trabajo.

El anexo A tiene como objetivo ilustrar cómo funcionan los algoritmos genéticos, describe el funcionamiento del algoritmo genético dando un ejemplo sencillo para diseñar una simple lata. Muestra la representación de cromosomas, función objetivo y los operadores de selección, cruzamiento y mutación en base al problema planteado, tal como lo describe D. E. Goldberg [Golberg 89, 91], Randy L. Haupt and Sue E. Haupt [Haupt 98] y Deb Kalyanmoy [Kalyanmoy 01].

El anexo B tiene como objetivo mostrar la dilatación, erosión, apertura y cierre, que implementan la teoría de conjuntos para dar origen al filtrado morfológico, tal como lo describe Douglas A. Lyon [Lyon 99], Rafael C. González y Richard E. Woods [González 95] y Gonzalo Pajares [Pajares 04].

Prologo.

Las interacciones 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) [OMG 01]. Ha habido importantes enfoques para extender funcionalidades de CORBA que soporten balanceo de carga, por ejemplo, TAO [Ossama 01a,b], VisiBroker [Lindermeier 00], MICO [Puder 02], 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 (RFP) [OMG 01b]. El RFP es una propuesta para extender la funcionalidad de CORBA para que consistente y eficientemente soporte balanceo de carga y servicio de monitoreo en ambientes basados en CORBA y sus aplicaciones. Tal extensión servirá de puente entre CORBA y otras metodologías de aplicación para procesar distribuidamente y en alto desempeño las aplicaciones que requieran estas características (balanceo de carga y monitoreo) y que actualmente no están implementadas en CORBA. Estas también agregaran capacidades de monitoreo y balanceo de carga a las aplicaciones basadas en CORBA.

El artículo de IONA Technologies [IONA 02] fue 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, por que, ella considera la información de estado actual para hacer decisiones de balanceo de carga.

Específicamente, la estrategia LL fue inicialmente propuesta por IONA [IONA 01]; TAO y PrimsTech también ofrecen implementaciones [Arapé 03]. 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 [Arapé 03], quien trata de ampliar LL para cubrir un sistema de computadoras heterogéneas; aunque no hay resultados reales sus simulaciones demostraron que es posible realizar este mejoramiento. La idea principal de este trabajo es desarrollar una estrategia para mejorar LL en un sistema de computadoras homogéneas y heterogéneas aplicando algoritmos genéticos, para balancear dinámicamente la carga. La estrategia se implemento en una arquitectura para balancear carga que fue desarrollada por nosotros bajo el estándar de CORBA y se evaluó con otras estrategias utilizando diferentes métricas de carga.

Resumen.

Los algoritmos de balanceo de carga pueden ser divididos en 2 grandes grupos: algoritmos de balanceo estático de *carga y* algoritmos de balanceo dinámico de *carga.* Los algoritmos de balanceo estático de *carga,* obtienen la localización de todos sus procesos antes de comenzar la ejecución. Los algoritmos de balanceo dinámico de *carga* intentan equilibrar la carga en tiempo de ejecución.

Este trabajo propone una nueva forma de balancear dinámicamente la carga bajo CORBA combinando una estrategia Genética en conjunto con Least_Loaded (LL), el trabajo se centra en la manera de combinar la estrategia Genética y LL con el propósito de mejorar el balanceo dinámico de carga. La estrategia se implementó en una arquitectura para balancear carga que fue desarrollada por nosotros bajo el estándar de CORBA, se evaluó con otras estrategias utilizando diferentes índices de desempeño (tiempo de respuesta global y rendimiento del sistema) y se ilustra cómo nuestra estrategia bajo ciertas condiciones del sistema arrojó mejores resultados.

Con el propósito de detectar las mejores condiciones del sistema se evaluaron las estrategias realizando balanceo dinámico de carga homogéneo y balanceo dinámico de carga heterogéneo en diversas aplicaciones. El balanceo dinámico de carga homogéneo fue aplicado utilizando un mismo tipo de requerimiento, arquitectura y velocidad de procesamiento. Una aplicación fue implementada cuyo objetivo es la transferencia de archivos, 4500, 9000, 18000, 27000 y 36000 archivos fueron evaluados con diferente granularidad (peso) y observamos que a medida que aumentamos la granularidad nuestra estrategia obtuvo mejores resultados, partiendo de esta observación el balanceo dinámico de carga heterogéneo solamente se evaluó con archivos de granularidad gruesa. Utilizamos diferentes sistemas operativos, arquitecturas y velocidades de procesamiento, una aplicación fue implementada cuyo objetivo fue la transferencia de archivos, 900 archivos con granularidad gruesa fueron evaluados. Cabe mencionar que no consideremos la misma cantidad de archivos por restricciones de almacenamiento.

Una vez detectadas las mejores condiciones del sistema implementamos dos aplicaciones con requerimientos de granularidad gruesa:

1. *Una aplicación para el procesamiento distribuido de imágenes fue implementada*. Evaluamos las estrategias: para implementar diversos operadores individuales (negativo, escala de grises, binarización y la potencia para abrillantar u oscurecer una imagen), para implementar el algoritmo de Zhang y Suen con el propósito de adelgazar (esqueletizar) los contornos de una imagen binaria y para implementar operaciones basadas en la convolución (evaluamos las estrategias implementado el operador Prewitt, Laplaceano y Robert) con el propósito de detectar bordes para delinear los límites de los objetos en la imagen.

2. *Una aplicación para la verificación de firmas off-line fue implementada.* El sistema de verificación fue basado sobre una arquitectura que opera en dos fases, la fase de entrenamiento y la fase de verificación. La fase de entrenamiento es formada por los estados de extracción de la firma, preprocesamiento y extracción de características y la generación del modelo. La fase de verificación

tiene etapas que son altamente similares a la fase de entrenamiento, en esta se comparten las tres primeras etapas y en la cuarta etapa las características extraídas de la firma son comparadas utilizando un clasificador de redes neuronales artificiales BPNN (Back-Propagation Neural Network) versus las características obtenidas en la fase de entrenamiento. Evaluamos nuestras estrategias en el sistema de verificación de firmas manuscritas *off-line* que fue implementado en nuestra arquitectura.

Al concluir nuestras evaluaciones y considerando el tiempo global y el rendimiento del sistema que son nuestros índices de desempeño, podemos decir que:

En el balanceo dinámico de carga homogéneo:

- Nuestra estrategia es una buena opción si la granularidad (peso) de los requerimientos es delgada y la demanda de los clientes es baja.
- Nuestra estrategia es una excelente opción si la granularidad de los requerimientos es gruesa y la demanda de los clientes es baja.
- Nuestra estrategia es la mejor opción si la granularidad es gruesa y la demanda de los clientes es alta.

En el balanceo dinámico de carga heterogéneo:

- Nuestra estrategia es la mejor opción si la granularidad heterogénea es gruesa y la demanda de los clientes es baja.
- Nuestra estrategia es la mejor opción si la granularidad heterogénea es gruesa y la demanda de los clientes es alta.

En el Procesamiento Distribuido de Imágenes.

- Nuestra estrategia es una excelente opción, cuando la granularidad es gruesa y la demanda del cliente es baja.
- Nuestra estrategia es la mejor opción, cuando la granularidad es gruesa y la demanda del cliente es alta.

En el Sistema de Verificación de Firmas *off-line*.

- Nuestra estrategia es una excelente opción, cuando la granularidad es gruesa y la demanda del cliente es baja.
- Nuestra estrategia es la mejor opción, cuando la granularidad es gruesa y la demanda del cliente es alta.

Capítulo 1. Componentes de CORBA .

La OMA (Object Management Architecture) define en alto nivel de abstracción las reglas necesarias para la distribución de la computación orientada a objetos (OO) en entornos heterogéneos. Uno de los elementos más importantes de la OMA es el ORB (Object Request Broker), el ORB es el encargado de dar transparencia en la comunicación a los clientes, en lo que se refiere al envío de requerimientos y al retorno de respuestas, cuando dichos clientes, solicitan los servicios de un objeto. Este capítulo tiene como principal objetivo describir de forma introductoria los principales componentes del ORB que son pieza fundamental de la arquitectura de CORBA tal como lo describe [Orfali 98, Vogel 98, López 01 y OMG 02].

1.1 Introducción.

CORBA es el resultado de la solicitud de un consorcio llamado Object Management Group (OMG) que es una organización internacional apoyada por más de 600 miembros, incluyendo vendedores de sistema de información, diseñadores de software y usuarios. Fundada en 1989, el OMG promueve la teoría y práctica de la tecnología orientada a objetos en el desarrollo del software [OMG 02]. CORBA es la especificación que permite integrar una gran variedad de sistemas de objetos. Con CORBA el cliente envía el requerimiento a la aplicación de objetos y ésta se encarga de realizar la operación. La importancia de CORBA radica en que permite definir la conexión prácticamente de cualquier forma existente de cliente/servidor. Los clientes simplemente invocan la operación o método del objeto del servidor y este se encargará de resolver dicho objeto contestando con el resultado. La magia de CORBA se basa en que la especificación del servidor es siempre independiente de su implementación en el cliente.

CORBA es una norma, no un producto. 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 (Object Request Broker). El ORB es la pieza fundamental de la arquitectura de CORBA denominada Arquitectura de Administración de Objetos ("*Object Management Architecture", OMA*). OMA define en alto nivel de abstracción las reglas necesarias para la distribución de la computación orientada a

objetos (OO) en entornos heterogéneos. OMA se compone de dos modelos, el *Modelo de Objetos* y el *Modelo de Referencia,* el primer modelo define cómo se deben describir los objetos distribuidos en un entorno heterogéneo y el segundo modelo caracteriza las interacciones entre dichos objetos.

La clave para entender la estructura de la arquitectura de CORBA es el Modelo de Referencia que está compuesto de cinco elementos principales (ver Figura 1.1) [Orfali 98]:

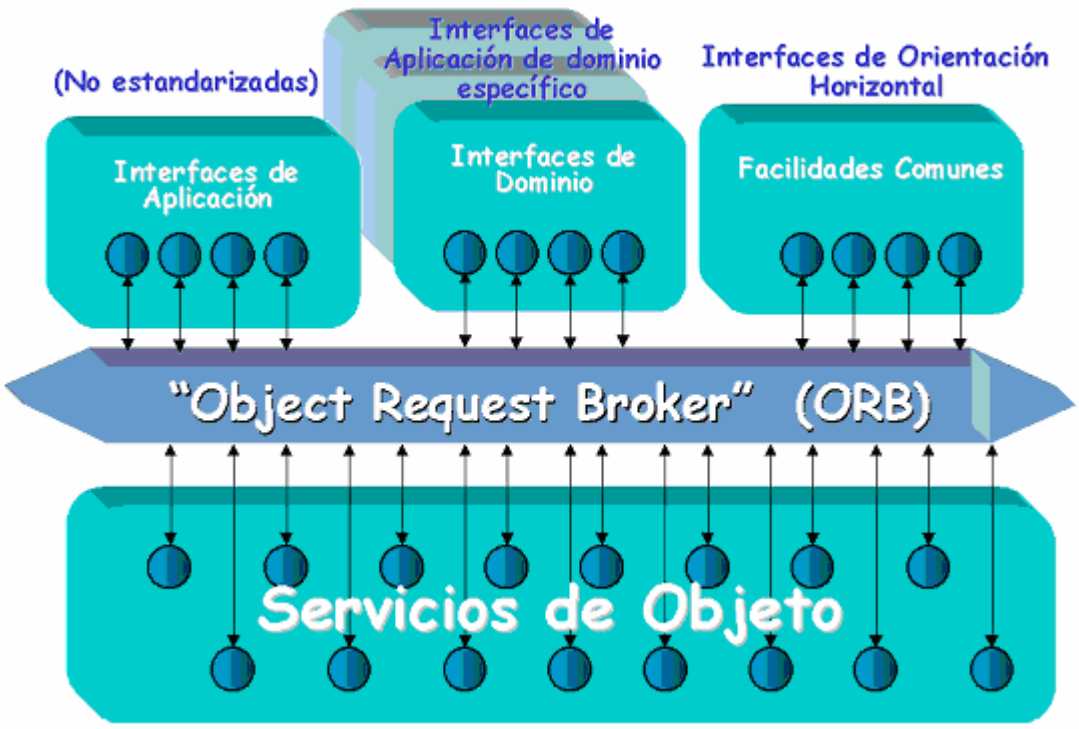


Figura 1.1 La Arquitectura para Manejar Objetos de la OMG [Orfali 98].

- ***Object Request Broker (ORB).*** El ORB es el encargado de dar transparencia en la comunicación a los clientes, en lo que se refiere al envío de requerimientos y al retorno de respuestas, cuando dichos clientes, solicitan los servicios de un objeto. El objeto que un cliente desea y al que el ORB envía sus requerimientos, es llamado el "*target object*". Mucha de la transparencia referida se ve reflejada en que comúnmente el ORB se encarga de la localización de los objetos ya que le cliente no conoce la implementación de los objetos con los que desea interactuar, ni el lenguaje de programación en que están escritos, ni el sistema operativo, ni el "hardware" sobre el cual están corriendo; el cliente tampoco se preocupa de la activación de los

objetos requeridos, ya que el ORB es el encargado de activar los objetos si fuese necesario, además, el cliente no necesitará conocer los mecanismos de comunicación (TCP/IP, llamada de métodos locales, etc.) que se utilizan, simplemente el ORB pasa los requerimientos de los clientes a los objetos y envía una respuesta a quien hizo el requerimiento; por otra parte, la transparencia del ORB permite que los desarrolladores se preocupen más de sus aplicaciones y menos de los asuntos que tengan que ver con programación de sistemas distribuidos a bajo nivel.

- ***Servicios de Objetos (Object Services).*** Son una colección de funciones básicas (interfaces y objetos) para usar e implementar los objetos. Es necesario construir servicios para cualquier aplicación distribuida que siempre sea independiente del dominio de la aplicación. Por ejemplo, el servicio de *Ciclo de Vida* define las funciones de crear, copiar, mover y eliminar objetos, pero no indica cómo los objetos se utilizarán en la aplicación.
- ***Facilidades Comunes (Common Facilities).*** Al igual que las interfaces de servicios de objetos, estas interfaces son de orientación horizontal (Es decir que pueden ser usadas en cualquier campo, por ejemplo en medicina, telecomunicaciones, tele-ingeniería, etc.) y están orientadas a aplicaciones de usuario final, como un ejemplo el *Distribute Document Component Facility (DDCF)* de OMG, basado en *OpenDoc de Apple Computer*, que permite la composición, presentación e intercambio de objetos basados en un modelo de documentos, facilitando por ejemplo, la incorporación de un objeto hoja de cálculo en un documento maestro de reportes (que es igualmente un objeto). Es posible entonces, que en un entorno de trabajo de personas geográficamente distribuidas, exista una aplicación que utilice el DDCF para la generación e intercambio de documentos. Las Facilidades comunes están en desarrollo permanente, entre ellas están, facilidades comunes para agentes móviles, intercambio de datos e internacionalización entre otras.
- ***Interfaces de Aplicación (Application Objects).*** Son los productos de un vendedor o grupo de desarrollo, para controlar las interfaces. Los objetos de la aplicación corresponden a la notación tradicional para aplicaciones, pero no son estandarizados por la OMG.
- ***Interfaces de Dominio.*** Estas interfaces al igual que las mencionadas anteriormente cumplen roles similares, excepto que las interfaces mencionadas en el presente contexto están orientadas a aplicaciones de dominio específico, por ejemplo, interfaces destinadas a aplicaciones financieras, o a aplicaciones de telecomunicaciones o de manufactura (como es el caso de uno

de los primeros *Request For Proposals* (*RFPs)* de *OMG* para la administración de datos de producto), es por ello que en la Figura 1.1 existen diferentes gráficas de dominios de interfaces, que representan un campo o un dominio de aplicaciones (en telecomunicación, medicina, etc.).

Para finalizar esta breve introducción a la OMA, cabe mencionar que existe un concepto llamado "*Object Framework*" o marco de objetos, la Figura 1.2 representa un conjunto de objetos que interactúan entre sí, formando una aplicación que soluciona un problema en un dominio específico, por ejemplo en telecomunicaciones, la banca, la manufactura o la medicina y cada uno de los círculos en la figura, representan componentes que se comunican entre sí a través de un ORB, los cuales soportan todas o algunas de las interfaces descritas en el *Modelo de Referencia* (Interfaces de Aplicación, Interfaces de Dominio, Facilidades Comunes y/o Interfaces de Servicios de Objetos) antes mencionado; estos componentes se comunican en forma "*peer-to-peer*), comportándose unas veces como cliente y otras como servidor.

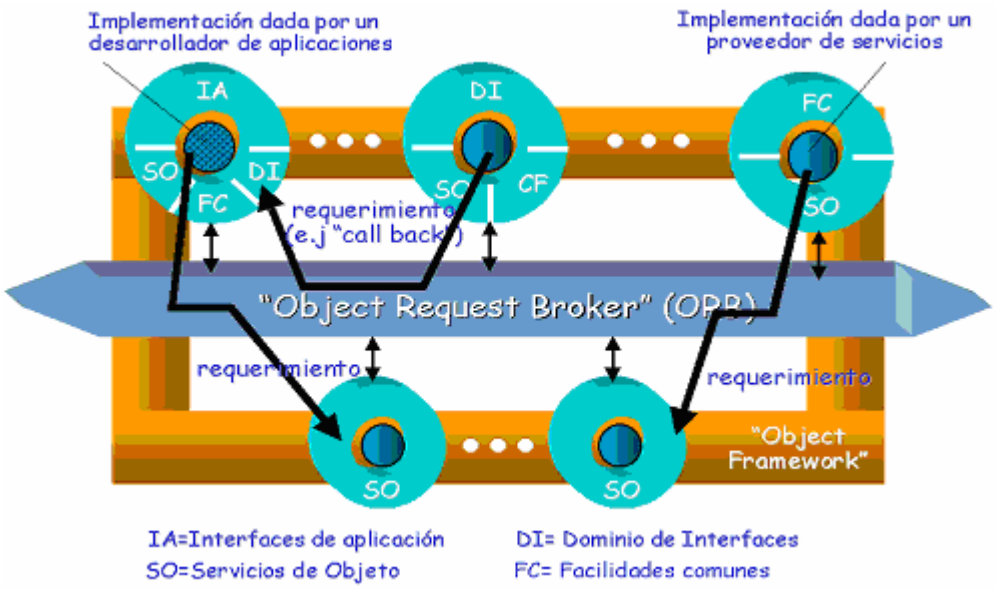


Figura 1.2 Especificación de un "Object Framework" en la OMA [Orfali 98].

Por otra parte, en la Figura 1.2 también se aprecia cómo el componente (Cliente) que soporta todas las interfaces, hace requerimientos a un componente (Servidor) que soporta solamente interfaces SO, así como otro componente cliente que soporta interfaces FC y SO, hace requerimientos a un componente servidor con interfaces SO. En la Figura 1.2, se observa que existe un requerimiento marcado como

"*Call-Back*", este caso se presenta cuando un componente que se comporta como un servidor, hace peticiones a un cliente, haciendo que dicho cliente pueda comportarse como cliente y servidor al mismo tiempo.

1.2 Anatomía del ORB de CORBA.

CORBA (*"Common Object Request Broker Architecture"*) es una infraestructura computacional abierta de objetos distribuidos, que ha sido especificada por la OMG, con el ánimo de describir todas las características del ORB de OMA [Orfali 98, Vogel 98 y OMG 02]. En la Figura 1.3 se puede apreciar cada uno de los componentes del ORB de CORBA, y a continuación se describe cada uno de ellos.

1.2.1 El Cliente.

Es la entidad que invoca operaciones sobre un objeto de *implementación.* Los servicios que brinda dicho objeto son transparentes, bastaría simplemente con invocar un método sobre un objeto; de tal forma que un objeto remoto para una entidad cliente se comporta como si fuese un objeto local (ver Figura 1.3).

1.2.2 Interfaz ORB.

Es un conjunto de librerías o *APIs* ("*Access Point Interfaces*") que definen un conjunto de funciones del ORB y que pueden ser accedidas directamente por el código cliente, entre ellas están las de convertir las *referencias de objetos* (cuando se solicita un servicio a un "*target object*" el servidor envía una referencia de dicho objeto, que en realidad es la información necesaria que un cliente necesita para inter-operar con el ORB y dicho "*target object*") en "*strings*" o viceversa y las que sirven para crear listas de argumentos de requerimientos, hechos a través de una invocación dinámica, vista un poco más adelante.

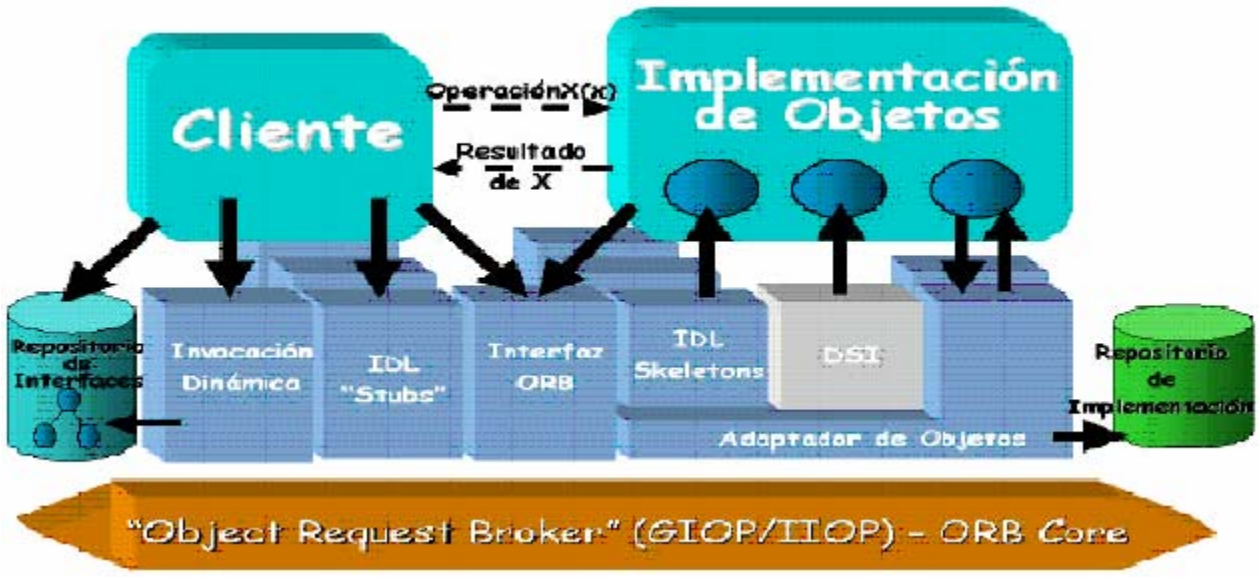


Figura 1.3 Componentes de CORBA [Orfali 98].

1.3 Lenguaje de Definición de Interfaces (IDL).

Cuando un cliente solicita los servicios de un objeto, este debe conocer las operaciones soportadas por dicho objeto, las interfaces de un objeto simplemente describen dichas operaciones. OMG IDL ("*Interface Definition Language*") es un lenguaje de "*especificación*" parecido en estructura a C++, que permite declarar el "contacto" de un objeto con el mundo exterior. Una de las ventajas de describir interfaces de esta forma, es separar los puntos de acceso a un objeto (sus interfaces) de su propia implementación, lo que permite que los objetos sean implementados en diferentes lenguajes de programación (C, C++, Java, Ada 95, SamallTalk, Cobol) e interactúen entre sí en forma transparente (aspecto importante en un sistema heterogéneo), ver Figura 1.4.

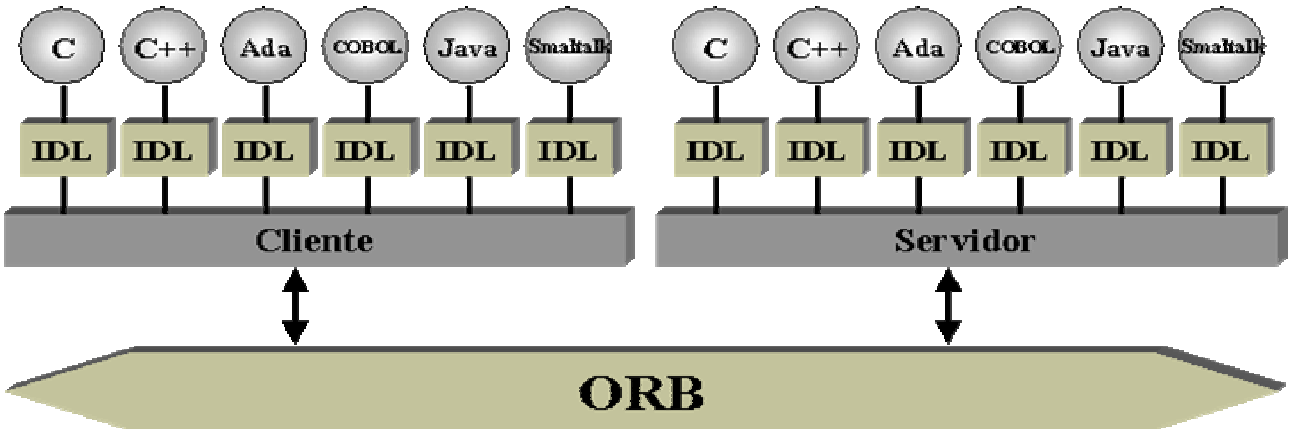


Figura 1.4 Interoperabilidad de Objetos Implementados en Diferentes Lenguajes [Orfali 98].

1.4 IDL "Stubs" e IDL "Skeletons".

Un "*Stub*" (normalmente llamado "*Prox*y") es un ente encargado de enviar los requerimientos de un cliente a un servidor a través del ORB (comúnmente llamado "*marshaling*", que consiste en convertir los requerimientos de un cliente implementado en algún lenguaje de programación en una representación adecuada para el envío de información a través del ORB); el "*Skeleton*" (en el servidor) es el encargado de colaborar con la recepción de dichos requerimientos desde el ORB y enviarlos a la Implementación de Objetos de CORBA (comúnmente llamado "*unmarshaling*", que es simplemente hacer una conversión de un formato de transmisión a un formato en un lenguaje de programación dado), visto de atrás hacia delante, a través del "*Skeleton*" se envía alguna respuesta a través del ORB y es recibida por el cliente por intermedio del "*Stub*", normalmente al conjunto de los envíos a través del "*Stub*" y el "*Skeleton*" es llamado *invocación estática* (tanto el cliente como el objeto de implementación, tienen pleno conocimiento de las interfaces IDL que están siendo invocadas).

1.5 Repositorio de Interfaces (IR).

El IR ("*Interface Repository*") es una base de datos distribuida que contiene información de las interfaces IDL definidas para los objetos que cooperarán en un entorno distribuido y que puede ser accedida o sobre escrita en tiempo de ejecución; podemos pensar en el IR como un Objeto CORBA, con una base de datos asociada y que tiene un conjunto de operaciones que pueden ser utilizadas como

si fuese un objeto cualquiera, entre los servicios que ofrece dicho Objeto CORBA, es permitir navegar sobre la jerarquía de interfaces almacenadas en la base de datos, de tal forma que se pudiese saber si se quisiera, la descripción de todas las operaciones que un objeto soporta. Una forma muy interesante y de mucha utilidad es usar el IR para descubrir interfaces de objetos en tiempo de ejecución, empleando invocación dinámica, que se verá a continuación.

1.6 La Interfaz de Invocación Dinámica y DSI.

El otro tipo de invocación que existe en CORBA es la *Invocación Dinámica*, que permite en tiempo de ejecución ("*run-time*"), descubrir las operaciones de un objeto, sin tener un conocimiento previo de sus interfaces (sin un "*stub*"). En la invocación dinámica existen dos tipos de interfaces, una es la Interfaz de Invocación Dinámica DII ("*Dinamic Invocation Interface*") y la otra es llamada del DSI ("*Dinamic Skeleton Interface*"); DII en una aplicación cliente se encarga de hacer peticiones de algún objeto del que no se conocen sus interfaces, dicha petición se hace a través de un pseudo objeto llamado "*request*", sobre el cual el cliente especifica el nombre de la operación y sus argumentos que pueden ser obtenidos del Repositorio de Interfaces(IR). Cuando el "*request*" esté completo se le envía al servidor, dicho envío puede hacerse de tres formas, en la primera el "*request*" se envía y todos los procesos se bloquean hasta que el servidor emita una respuesta (Invocación Sincrónica), en la segunda cuando el "*request*" se envía, el cliente sigue procesando y más tarde recoge la respuesta (Invocación Sincrónica Aplazada), en la última forma cuando se envía el "*request*", el cliente sigue procesando y la respuesta del servidor se recoge por algún otro medio (por ejemplo un proceso separado que la recoja). En el lado del servidor, cuando un "*request*" (pseudo objeto) es recibido, el DSI es quien lo toma y envía alguna respuesta al cliente ante la petición solicitada (lo que es llamado el "*Dispatching*"). En la Figura 1.5 se puede apreciar de una manera gráfica una forma de hacer invocación dinámica.

1.7 El Adaptador de Objetos.

Es el ente de contacto entre el ORB y los objetos de implementación y es quien acepta requerimientos en nombre de los objetos servidores, dicho adaptador se encarga en tiempo de ejecución de activar, instanciar, pasar requerimientos y generar referencias de dichos objetos. También, de colaborar con el ORB para que todos los requerimientos que se hagan de múltiples conexiones, sean recibidos sin

ningún tipo de bloqueo. El adaptador de objetos tiene tres interfaces asociadas, una al DSI, una al IDL "*skeleton*" y otra a la implementación de objetos (Ver Figura 1.3) siendo las dos primeras privadas y la última pública, todo esto con el ánimo de aislar la implementación de los objetos del ORB tanto como sea posible. La OMG estandarizó un adaptador de objetos llamado BOA ("*Basic Object Adaptator*"), dicho BOA y los servidores, tiene la posibilidad de soportar más de un adaptador de objetos. OMG actualmente ha lanzado una especificación que mejora algunos defectos de portabilidad del adaptador de objetos BOA y es llamada POA *("Portable Object Adaptator*").

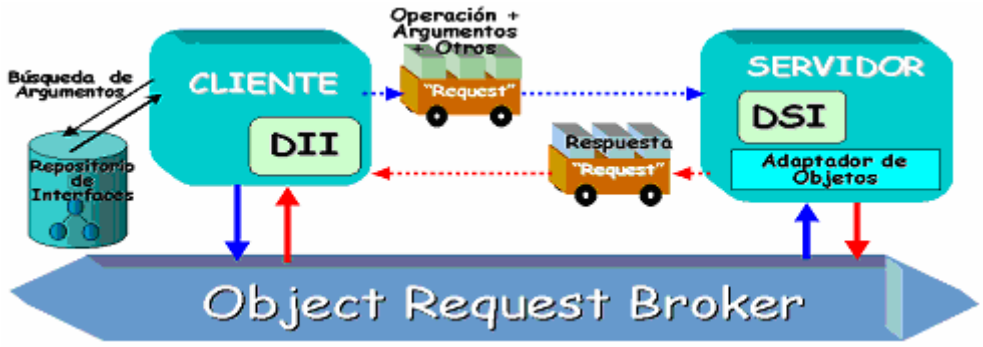


Figura 1.5 Invocación Dinámica con DII y DSI [Orfali 98].

1.8 Repositorio de Implementación.

Es una base de datos que en tiempo de ejecución, da información acerca de las clases que un servidor soporta, los objetos que están instanciados, sus identificadores "*IDs*" (es un número único que asigna el adaptador de objetos a cada instancia de un objeto) y una serie de datos administrativos de los objetos, como trazas de información e información de seguridad entre otros.

1.9 Protocolos del ORB (GIOP/IIOP).

CORBA tiene una arquitectura general de interoperabilidad entre ORBs, una directa ORB a ORB y otra basada en puentes ("*Bridge*"), el primer tipo de interoperabilidad es dada cuando los ORBs residen en el mismo dominio, compartiendo las mismas referencias de objeto, el mismo tipo de información IDL y tal vez la misma información de seguridad; el segundo tipo de interoperabilidad se da cuando se desean comunicar ORBs de diferentes dominios, entonces el "Bridge" se encarga de

mapear la información específica de un ORB a otro, en la Figura 1.6 se puede apreciar esta característica. La arquitectura de interoperabilidad de ORBs es basada en GIOP ("*General Inter-ORB Protocol*"), que especifica un conjunto de formatos de mensajes y representaciones de datos para la interacción entre ORBs; GIOP es diseñado para trabajar sobre cualquier protocolo de transporte orientado a conexión, por ejemplo el protocolo IIOP ("*Internet Inter-ORB Protocol*") especifica como mensajes GIOP son intercambiados sobre redes TCP/IP, gracias a IIOP, es posible usar Internet como un *Backbone* ORB sobre el cual, otros ORBs pueden conectarse. (Para ser compatibles con el ORB de CORBA se debe soportar GIOP sobre TCP/IP).

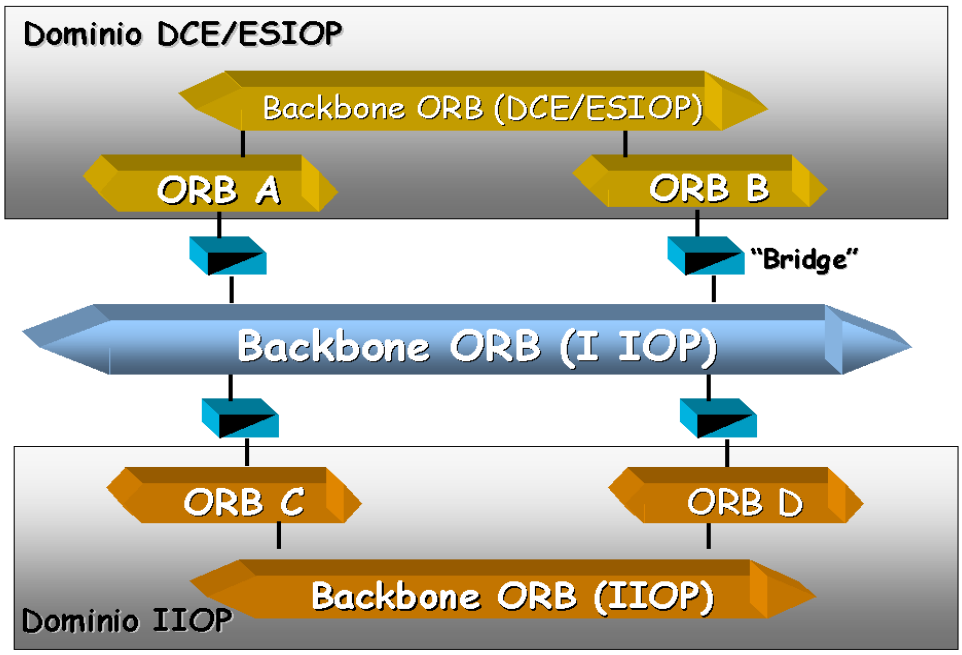


Figura 1.6 Federación de ORBs [Orfali 98].

Por otra parte la arquitectura de interoperabilidad entre ORBs, define un conjunto de protocolos llamados ESIOPs ("*Environment-specific inter-ORB Protocols*") que hacen posible la interacción de ORBs sobre redes específicas, por ejemplo uno de los primeros ESIOPs especificados fue el utilizado por el entorno de computación distribuida DCE ("*Distributed Computing Environment*") llamado DCE-CIOP ("*DCE Common Inter-ORB Protocol*"), con el ánimo de que el mundo de las aplicaciones CORBA y DCE, interoperen en forma transparente (Ver Figura 1.6), es importante destacar aquí, que GIOP especifica un formato para las referencias de los objetos, llamado IOR ("*Interoperable Object*

Reference"), dicho IOR almacena información necesaria para localizar y comunicar un objeto sobre uno o más protocolos, como datos que identifiquen el dominio del ORB sobre el cual una referencia es asociada y también los protocolos que ésta soporta, por ejemplo un IOR que contiene información de un dominio IIOP almacenara información de un nombre de un "*Host*" (Nombre de una máquina o dirección IP) y un número de puerto TCP/IP.

1.10 Conclusiones.

CORBA es el resultado de la solicitud de un consorcio llamado Object Management Group (OMG) que es una organización internacional apoyada por más de 600 miembros, incluyendo vendedores de sistema de información, diseñadores de software y usuarios. Fundada en 1989, la OMG promueve la teoría y práctica de la tecnología orientada a objetos en el desarrollo del software [OMG 02]. La magia de CORBA se basa en que la especificación del servidor es siempre independiente de su implementación en el cliente. CORBA es una norma, no un producto. CORBA se encarga de especificar, en un entorno distribuido heterogéneo, el intercambio de operaciones entre objetos de manera transparente. Este capitulo describió de forma introductoria los principales componentes de CORBA, objetivo y función, mas sin embargo no pretende ser un manual de referencia de la arquitectura de CORBA, una descripción mas detallada se puede obtener en [Orfali 98, Vogel 98, López 01 y OMG 02].

Capítulo 2. Taxonomía de los Algoritmos de Balanceo de Carga.

Este capítulo ilustra una taxonomía de los algoritmos de balanceo de carga tal como lo describe Casavant y Kuhl [Casavant 88a], la taxonomía describe varias clases de algoritmos en base a un esquema de clasificación jerárquica, los algoritmos que no pueden ser clasificados jerárquicamente son clasificados en forma plana (flat). El capitulo resalta la diferencia que existe entre las políticas que mantienen su estado de ejecución con las políticas que no mantienen su estado de ejecución de acuerdo con Hisao Kameda en [Kameda 97]. El capitulo concluye mostrando varias metas que se siguen en la distribución de carga tal como lo mencionan Mukesh Singhal y Niranjan G. Shivaratri [Singhal 94], A. Goscinski [Goscinski 94], Hisao Kameda [Kameda 97] y Behrooz A.Shirazi [Shirazi 95].

2.1 Definición de Balanceo de Carga.

El *balanceo de carga* es parte del amplio problema de asignación de recursos. El problema de *balanceo de carga* (llamado algunas veces planificación distribuida) es cómo distribuir procesos entre los elementos de procesamiento conectados por una red para equilibrar la carga de trabajo entre ellos y lograr algunas metas de desempeño tales como: minimizar el tiempo de ejecución, minimizar los retardos de comunicación, y/o maximizar la utilización de recursos [Goscinski 94]. Mientras la compartición de carga usa migración de procesos solamente cuando hay un procesador ocioso, el balanceo de la carga debe de requerir migración aun cuando un nodo no este ocioso. Antes de entrar de lleno en el balanceo de carga, se deben clarificar tres términos básicos que son de uso común en la literatura: balanceo de carga, planificación distribuida y asignación de recursos. A menudo existe una distinción implícita entre esos conceptos, pero la pregunta sería, ¿existe alguna distinción real entre ellos?.

Dando un vistazo al problema de manejo de recursos propuesto por Casavant y Kuhl 1988a [Goscinski 94]. Se Recuerda que el manejo de recursos es una política para acceder y usar recursos efectiva y eficientemente por los clientes (procesos y objetos activos). Esto es, en este problema se pueden distinguir los siguientes componentes: clientes, recursos y políticas. La relación entre ellos es ilustrada en la Figura 2.1.

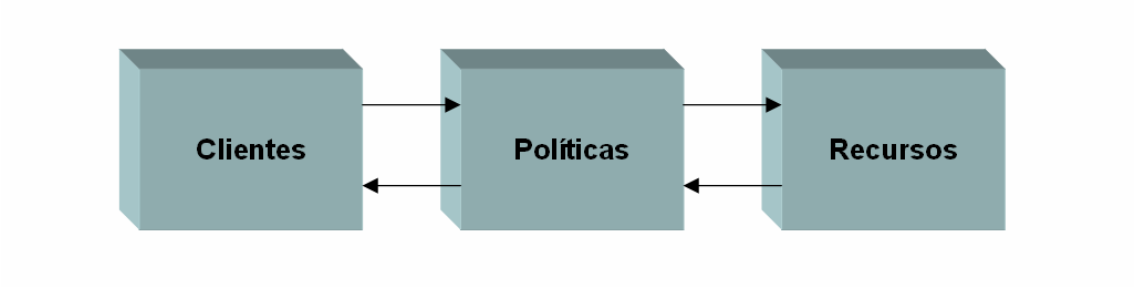


Figura 2.1 Las Relaciones Entre Clientes, Políticas y Recursos en el Problema de Manejo de Recursos [Goscinski 94].

Si se observa el comportamiento del sistema ilustrado en la Figura 2.1 desde el punto de vista de los recursos, se ve el problema de *asignación de recursos*. Por otra parte, el comportamiento del sistema desde el punto de vista de los clientes lleva al problema de planificación, en ambientes distribuidos se le conoce como el problema de *planificación distribuida*. La asignación de recursos y la planificación son dos términos que describen el mismo problema de manejo de recursos pero de dos puntos de vista diferentes.

La *asignación de recursos* esta principalmente envuelta en la planificación de trabajos y recursos periféricos en el procesador. La *planificación distribuida* hace referencia a un conjunto de políticas y mecanismos que goviernan el orden en que se ejecutan los trabajos. Un planificador es un mecanismo que selecciona el siguiente trabajo que hay que admitir en el sistema y el siguiente trabajo que hay que ejecutar [Milenkovic 94].

El *balanceo de carga* es la parte de la política de planificación distribuida que es responsable de la distribución de la carga del sistema entre los elementos de procesamiento a través de la migración de procesos. Esto es, el *balanceo de carga* fuertemente refleja los requerimientos del cliente. Sin embargo, describe el hecho de que balancear la carga es un término reducido más que planificación distribuida, por que este refleja y cubre diferentes enfoques. En este escrito usaremos balanceo de carga y planificación distribuida indiferentemente.

2.2 Clasificación Jerárquica de los Algoritmos de Balanceo de Carga.

La clasificación jerárquica de los algoritmos de balanceo de carga es ilustrada en la Figura 2.2 [Casavant 88a]). Se discuten varias clases de algoritmos siguiendo esta clasificación: Como la planificación local es descrita en numerosas referencias [Milenkovic 98], [Silberschatz 04] por citar algunas de ellas, se centrará en la planificación global (balanceo de carga).

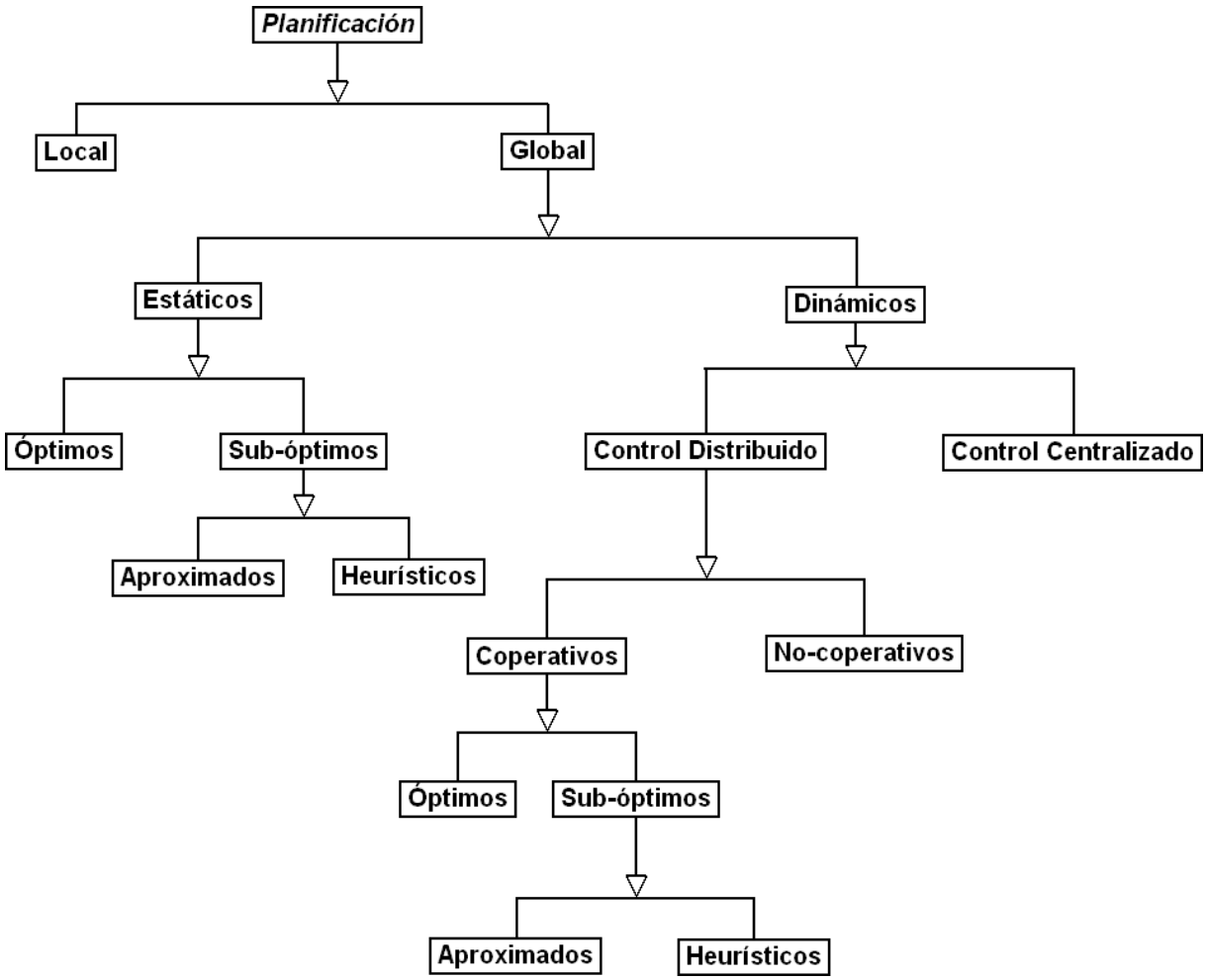


Figura 2.2 Clasificación Jerárquica de los Algoritmos de Balanceo de Carga (Adaptada por Casavant y Kuhl [Casavant 88a])

Los algoritmos de balanceo de carga deben ser divididos en dos grandes grupos: estáticos y dinámicos. Esta división esta basada en el tiempo en que se hacen las decisiones de balanceo de carga. Primero se

describen los algoritmos de balanceo de carga estáticos y enseguida los algoritmos dinámicos.

2.3 Algoritmos de Balanceo Estático de Carga.

En la planificación estática 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 [Shirazi 95]. Típicamente las metas de los métodos de planificación estáticos es minimizar el tiempo de ejecución global para un programa concurrente, mientras minimizamos los retardos de comunicación. Con esta meta en mente, los métodos de planificación estáticos intentan [Shirazi 95]:

- Predecir el comportamiento del programa en ejecución en tiempo de compilación (esto es, estima los procesos o tareas, tiempos de ejecución y retardos de comunicación).
- Procesar el particiónado de pequeñas tareas en procesos de grano grueso en un intento por reducir los costos de comunicación.
- Asignar procesos a procesadores.

La mayor ventaja de los métodos de planificación estáticos es que toda la sobrecarga (overhead) de la planificación de procesos es conocida en tiempo de compilación, resultando un tiempo de ejecución más eficiente comparado con los métodos de planificación dinámicos. Sin embargo los métodos de planificación estáticos sufren de muchas desventajas, como discutiremos posteriormente.

2.3.1 Óptimos Vs. Sub - Óptimos.

El balanceo de carga estático es simplemente un problema matemático, que corresponde al estado del sistema y los requerimientos de recursos contrastando con algún índice de desempeño (función de criterio). Las siguientes medidas son de uso común:

1) Minimizar el tiempo de procesamiento global.
2) Maximizar la utilización de recursos.
3) Maximizar el rendimiento del sistema (throughput).

Tal vez una de las más críticas limitaciones de la planificación estática es que, en general, la planificación óptima genera un problema NP-completo. Problemas NP-completos, para métodos de planificación estáticos, óptimos, globales, con o sin consideraciones de costos de comunicación, han sido considerados en la literatura en Chou 1983, Ma 1982 y Shen 1985 [Shirazi 95]. Esos algoritmos han sido desarrollados basados sobre teorías de colas, programación matemática y teorías de grafos.

El tamaño de un sistema distribuido típico (un gran número de procesos, procesadores y otros recursos que imponen algunas restricciones) puede crecer de tal forma que el balancear carga estáticamente lleva a un problema de computación complejo. Esto es obtener soluciones óptimas puede ser muy caro y en muchos casos no confiables en un periodo de tiempo razonable. Esto lleva al uso de soluciones sub – óptimas. Hay dos categorías de soluciones sub – óptimas: aproximadas y heurísticas.

2.3.2 Aproximados Vs. Heurísticos.

El enfoque aproximado usa el mismo modelo matemático y computacional para el algoritmo, pero busca una solución que no cubre todo el espacio de solución. El espacio de solución es buscado en profundidad primero o en amplitud primero. Esto significa que la meta no es obtener una solución óptima sino solamente una satisfactoria, buscar solamente una buena solución minimiza el tiempo. Esta es la principal ventaja de esta clase de algoritmos. El problema es cómo determinar que la solución es lo suficientemente buena.

Los siguientes factores determinan si este enfoque es digno de usarse:

1) Disponibilidad de una función para evaluar una solución.
2) El tiempo requerido para evaluar la solución.
3) Habilidad para evaluar de acuerdo alguna métrica el valor de alguna solución óptima.
4) Habilidad de un mecanismo para que inteligentemente detecte el espacio de solución.

Algoritmos de balanceo de carga aproximados, sub – óptimos, estáticos y globales han sido descritos por Bannister-Trevi 1983 y Lo 1984 [Goscinski 94]. El algoritmo formal es basado sobre programación matemática y posteriormente sobre teoría de grafos.

La segunda categoría de algoritmos sub – óptimos es basada sobre estrategias de búsqueda heurísticas. Los métodos heurísticos como su nombre lo indica, relaciona reglas thumb para guiar el proceso de planeación en la dirección correcta y alcanzar una solución óptima (o cerca a la óptima). Una de las más importantes características de estos algoritmos son las suposiciones acerca del conocimiento a priori de procesos y la carga del sistema que es más realista; ya que este tipo de algoritmos hacen uso de parámetros especiales que afectan al sistema en forma indirecta. A menudo, los parámetros que serán monitoreados son correlacionados en forma indirecta con el desempeño del sistema y este parámetro alterno es mucho más simple de calcular y monitorear. El tiempo de computación y la cantidad de recursos requerida para obtener soluciones basadas sobre estos algoritmos es razonable. Algunas estrategias de balanceo de carga heurísticas, sub–óptimas, estáticas y globales son descritas en Efe 1982 [Goscinski 94].

Ambas soluciones óptimas y sub-óptimas – aproximadas son basadas sobre el mismo modelo de computación formal. Esos modelos pueden ser desarrollados basados sobre las siguientes áreas: programación matemática, teoría de grafos, teoría de colas y soluciones por espacio de enumeración-búsqueda. No se analizarán o presentarán algunos de los algoritmos que corresponden al área estática en la taxonomía jerárquica ya que ellos no tienen un buen desempeño en sistemas de computación distribuida. Esto es simplemente por que ellos no toman en consideración el estado actual del sistema y la fluctuación de carga de trabajo.

2.3.3 Pronóstico y Futuro en la Planificación Estática.

Shirazi concluye que hay un gran número de métodos de planeación estática de alta calidad disponibles en la literatura que son sumamente erróneos en aplicaciones prácticas [Shirazi 95]. Este hueco implica que se pudiera necesitar todo un conjunto de herramientas y técnicas que realicen el puente entre los resultados de búsqueda actual en esta área y sus aplicaciones prácticas. Algunas de las áreas activas actuales así como las investigaciones futuras y direcciones de desarrollo en la planeación estática son:

El desarrollo de una herramienta generadora de Grafos Acíclicos Dirigidos "generador de DAG ".

La entrada de un generador DAG será un programa paralelo, escrito en algún lenguaje y la salida será el programa equivalente DAG. Actualmente, existen herramientas de planeación usando código especializado para la generación de DAG o cuentan con un lenguaje funcional, tal como SISAL, para una fácil conversión de los programas a una forma DAG. Sin embargo, la mayoría de aplicaciones industriales y comerciales son desarrolladas en C, Fortran, o una variación cercana a tales lenguajes. Esto es, para el éxito práctico de los métodos de planeación, es indispensable desarrollar una herramienta que pueda generar DAG de C o código Fortran.

El desarrollo de una herramienta para "estimar el tiempo de ejecución de las tareas". Una de las más importantes características de planeación estática es la suposición de que los tiempos de ejecución de las tareas son conocidos antes de la ejecución actual de las tareas. Esta suposición es a menudo no realista dado que los tiempos de ejecución de las tareas pudieran variar debido a los condicionales y construcciones de ciclos y las variaciones en los valores de entrada. También, las estimaciones exactas de retardos de comunicación son difíciles, debido al comportamiento dinámico de los sub-sistemas de red. Esto es una herramienta generadora de DAG debe ser aumentada con una herramienta que pueda proveer razonablemente estimaciones exactas de los tiempos de ejecución de las tareas y los retardos de comunicación en tiempo de compilación. Varias direcciones pueden ser investigadas en esta área, tales como:

- *Estimaciones de Usuario.* Los programadores anotarán el programa con sus estimaciones de los tiempos de ejecución. En un método relacionado, los programadores pudieran pedir especificar la probabilidad tomando una condición, o estimando el número de condiciones en un ciclo. Tal información puede entonces ser usada para producir estimaciones probabilísticas de los tiempos de ejecución de las tareas.
- *Estimaciones Basadas en Simulación (Simulation-Based).* El comportamiento de un programa en tiempo de corrida puede ser simulado, en tiempo de compilación, ejecutando las instrucciones que afectan el flujo de control del programa. Para las instrucciones restantes, el tiempo simulado es deducido en consecuencia. Las estadísticas respaldan la frecuencia de ejecución de las instrucciones y sus retardos de ejecución estimados, que puedan ser asociados de esta manera.
- *Estimaciones Basadas en Perfiles (Profiled-Based).* La idea es monitorear corridas previas

del programa, o secuencialmente ejecutar el programa y coleccionar los retardos de tiempo de ejecución y frecuencia de ejecución de las tareas.

Desarrollo de una Herramienta para el Desempeño de Perfiles (Performance Profiler). La función de una herramienta de desempeño de perfiles es leer en la planeación de un programa en paralelo sobre alguna arquitectura dada y producir un perfil gráfico del desempeño esperado del programa de entrada. La herramienta debe también proveer un "perfil de paralelismo ideal" para la aplicación, dando al usuario una idea de paralelismo inherente en el programa.

Desarrollo de una Herramienta de Distribución de Datos. Los métodos de planeación existentes son basados simplemente sobre la función de distribución (planeación), ignorando la meta de distribución de datos. Esta omisión causa degradaciones de desempeño, debido a los retardos de comunicación en tiempos de corrida para acceder sitios de datos remotos. Algunos métodos de planeación funcional deben ser aumentados con una herramienta de distribución de datos, en el cual los datos necesarios para planear las particiones del programa pudieran ser accesados localmente en la mayoría de las veces.

2.4 Algoritmos de Balanceo Dinámico de Carga.

En el balanceo de carga estático, las decisiones con respecto a la localización de los procesos son hechas antes de que ellas comiencen a ejecutarse. Esas decisiones son basadas sobre suposiciones acerca de los procesos que serán ejecutados (tiempo de ejecución), ambiente de ejecución (carga actual, tiempo de transferencia y costos de comunicación). En la práctica, esta información es usualmente no conocida, solamente muy poco conocimiento a priori esta disponible.

El balanceo de carga dinámico intenta ecualizar la carga del sistema dinámicamente cuando los procesos son creados o reactivados. Esos algoritmos permiten a los procesos migrar a computadoras remotas una vez que ellos hayan comenzado su ejecución. La meta de esta migración es obtener la mejor localización para todos los procesos existentes en un sistema distribuido. Lo mejor aquí es, que la migración es usada en base a índices de desempeño y restricciones. Las decisiones son hechas usando la información sobre el estado del sistema actual, esto es, cada computadora debe conocer el

estado de otras computadoras.

Obtener una solución dinámica es mucho más complicado que obtener una solución estática. Esto requiere agrupamiento y mantenimiento de estado de información. Sin embargo la asignación dinámica de carga puede obtener un buen desempeño haciendo decisiones de migración de procesos utilizando índices de carga que chequen el estado del sistema actual. Un problema claramente asociado con el agrupamiento y mantenimiento de estado de información es ¿dónde las decisiones serán hechas?.

Kameda [Kameda 97] y Goscinski [Goscinski 94] señalan que los algoritmos dinámicos de balanceo de carga consisten de tres componentes básicos: (1) una *política de información* que es responsable de coleccionar información de estado del sistema, (2) una *política de transferencia* que determina si un nodo está en un estado adecuado para participar en la transferencia de una tarea, (3) una *política de localización* que determina a qué nodo una tarea seleccionada deberá de ser enviada (Figura 2.3).

1) *Política de Información*. La política de información es responsable de decidir ¿cuándo la información de estado de otros nodos debe de ser coleccionada?, ¿dónde debe de ser coleccionada? y ¿qué información debe ser coleccionada?. La mayoría de las políticas de información usan uno de los siguientes dos tipos [Carretero01]:
   Bajo Demanda. La información se recupera sólo cuando un nodo se convierte en un emisor o receptor de procesos. La política puede ser inicializada por el emisor o inicializada por el receptor. En el primer caso es la computadora emisora la encargada de buscar posibles nodos receptores del proceso. En el segundo caso es la computadora receptora (cuando su carga es baja) la que solicita procesos a otros nodos.
   Periódicas. Los nodos intercambian información periódicamente. Este tipo de políticas introduce una sobrecarga constante en el sistema y no se adaptan a las necesidades del mismo. Así, por ejemplo, si la información se intercambia con una frecuencia muy baja, puede haber un reparto de la carga muy ineficaz. Si por lo contrario, el algoritmo intercambia información con mucha frecuencia, puede introducir una carga excesiva en el sistema.
2) *Política de transferencia*. Un gran número de políticas de transferencia han sido

propuestas como políticas umbral (T). Los umbrales son expresados en unidades de carga. Cuando una nueva tarea se origina en un nodo, y la carga en ese nodo excede el umbral T, la política de transferencia decide que el nodo es un emisor. Si la carga en el nodo cae por debajo del umbral T, la política de transferencia decide que el nodo puede ser un receptor para una tarea remota. La política de transferencia inicia la transferencia de tareas cuando se detecta que la carga entre nodos no esta balanceada debido a las acciones tomadas en la política de información.

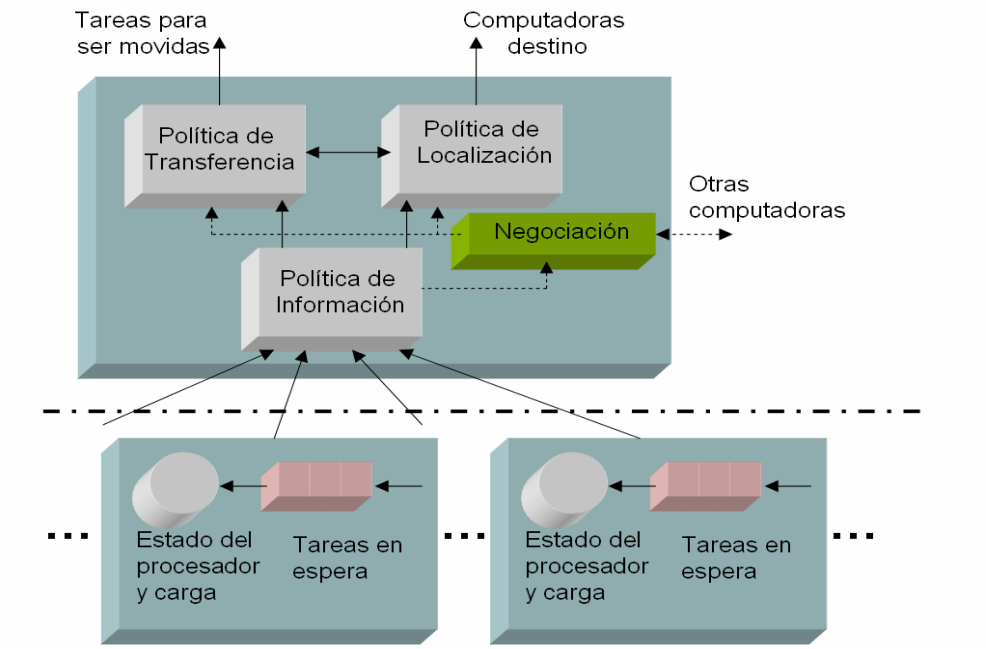


Figura 2.3 Componentes de un Algoritmo Dinámico de Balanceo de Carga.

3) *Política de localización*. La política de localización determina a que nodo una tarea seleccionada deberá transferirse. La política de localización puede ser: 1) sin estado de información, no usa información de estado de otros nodos en el sistema y, 2) con estado de información, se realiza un análisis de estado de un nodo y se checa si en este nodo se puede compartir la carga.

Componente de negociación. Los tres componentes de los algoritmos de balanceo de carga distribuidos no están aislados de los componentes lógicos de otras computadoras. Ellos interactúan de

varias formas, en general a través de los componentes de negociación.

2.4.1 Control Distribuido Vs. Control Centralizado.

Esta meta involucra si la información de estado de todos los procesadores y ambientes de ejecución será coleccionada en alguna localización o si las decisiones que se hacen son físicamente distribuidas entre procesadores que utilizan información almacenada en diferentes lugares.

La más importante característica para hacer decisiones centralizadas es su simplicidad. Sin embargo tales sistemas sufren de un número de desventajas. La primera desventaja es que agrupan información y mantienen el estado del sistema en una localización centralizada que puede conducir a una gran sobrecarga (overhead) dado que hay retardos de transferencia y mensajes que pudieran estar perdidos. Este es un problema muy serio ya que es bien conocido de la teoría de optimización que las decisiones basadas sobre la información que no esta actualizada puede ser peor que ninguna decisión. La segunda desventaja es la baja confiabilidad de tales sistemas, las fallas del nodo en que las decisiones de balanceo de carga se hacen colapsan el sistema entero. Algoritmos de este tipo son discutidos por Chow and Kohler 1979 y Ousterhout 1980 [Goscinski 94].

En sistemas de asignación de carga distribuida las decisiones que se toman son físicamente distribuidas entre los procesadores, en este tipo de sistemas no se tienen los problemas de los sistemas de control centralizado. Los cuellos de botella para seleccionar información de estado en un solo lugar son evitados, si una computadora falla otras pueden continuar su trabajo. Algoritmos de este tipo son discutidos por Enslow 1978 y Eager D., Lazowska E., and Zahorjan J. 1986 [Casavant 88a].

2.4.2 Cooperativos Vs. No–Cooperativos.

En un sistema de distribución carga dinámico, muchos componentes están envueltos y son responsables de tomar decisiones. Si esos componentes distribuidos cooperan en tomar decisiones, convenimos con algoritmos cooperativos. Todas las computadoras trabajan hacia una meta común pensando que cada una de ellas es responsable de una porción en la planificación de tareas. En la clase de algoritmos cooperativos es posible identificar los llamados algoritmos bidding (ofrecimiento). El

concepto básico en este grupo es la negociación entre componentes, sometiendo bids (ofrecimientos) a contratos, ejemplos de este tipo de algoritmos los podemos encontrar en Casavant and Kuhl 1984, Ramamritham and Stankovic 1984, Stankovic and Sidhu 1984 en [Goscinski 94].

Por otra parte, si las computadoras toman decisiones independientemente una de la otra, el algoritmo es llamado no-cooperativo. En este tipo de algoritmo los componentes actúan como entidades autónomas que son orientadas solamente hacia metas individuales. Como resultado, sus decisiones solamente afecta al desempeño local, ejemplos de este tipo de algoritmos los podemos encontrar en Klappholtz and Park 1984 y en Reif and Sprirakis 1984 en [Goscinski 94].

2.5 Taxonomía Plana de los Algoritmos de Balanceo Dinámico de Carga.

Algunos descriptores de los algoritmos de balanceo de carga no pueden ser escogidos en forma jerárquica, ya que algunos algoritmos se centran en seleccionar una computadora remota, que puede ser la computadora destino para migrar los procesos, otro tipo de algoritmos tales como el adaptativo modifica dinámicamente el algoritmo y los parámetros para implementar la política de planificación de acuerdo al comportamiento previo y al actual del sistema escogiendo los descriptores del sistema en forma arbitraria, esta fue la principal razón para el desarrollo de la clasificación plana (flat) propuesta por Casavant and Khul 1988 [Casavant 88a].

2.5.1 Adaptativos Vs. No-Adaptativos.

Esta clasificación es asociada con la parte de datos usada para tomar una decisión, o bien si solamente la información del estado actual es usada o el comportamiento previo del sistema junto con el estado actual serán usados para tomar decisiones. Un planificador adaptativo es el que modifica dinámicamente el algoritmo y los parámetros para implementar la política de planificación de acuerdo al comportamiento previo y al actual del sistema. Esto significa que se basa sobre la historia de la actividad del sistema. Note que esta planificación es también llamada probabilística. Stankovic-Sidhu 1984 en [Goscinski 94], desarrollaron un algoritmo de planificación que considera muchos parámetros cuando se toman decisiones. El planificador observa la respuesta del sistema para inicializar los valores de los parámetros. Como resultado, éste ignora algunos de esos parámetros si el

planificador piensa que no proveen información acerca del estado del sistema.

2.5.2 Ofrecimiento (Bidding).

Bidding es un enfoque usado para seleccionar una computadora remota, que puede ser la computadora destino para migrar un proceso. El Bidding es ejecutado cuando una computadora no tiene información exacta acerca del estado de otras computadoras, o información que no tiene para permitir identificar una buena computadora destino para un proceso seleccionado. El concepto detrás del enfoque bidding es la negociación entre componentes para un sistema de balanceo de carga (agentes cooperativos) y someter ofrecimientos (bids) a contratos.

La negociación bidding tiene cuatro características.

1) Es un proceso local que no involucra control centralizado.
2) La información es intercambiada en ambas direcciones.
3) Cada componente de negociación par a par evalúa la información desde su propia perspectiva y usualmente es basada sobre un surplus (sobrante) en los recursos necesitados por un proceso para ser migrado.
4) Un acuerdo final es logrado por selección mutua.

Cada computadora en un sistema distribuido es responsable de una de las dos funciones en los procesos bidding:

1) Manejador (también llamado agente local). Este trabaja en nombre de un proceso, intentando obtener una localización para su ejecución.
2) Contratista (también llamado agente remoto). Representa una computadora remota que está disponible para trabajar un proceso de otra computadora.

Las computadoras no son diseñadas a priori como los manejadores o los contratistas. La función puede ser cambiada dinámicamente cuando se hacen decisiones. Una computadora puede ejecutar ambas funciones simultáneamente para diferentes contratos. Los sistemas bidding trabajan en la siguiente

forma:

1. El manejador anuncia la existencia de algún proceso que será ejecutado mediante un broadcasting o direcciona un mensaje request for bid. Al mismo tiempo ésta computadora puede recibir mensajes de otras computadoras.
2. Los contratistas evalúan los mensajes de otras computadoras y remiten bids para que ellas sean ubicadas.
3. El manejador evalúa los bids recibidos y premia los contratos de los nodos que determina sean los más apropiados.

De acuerdo a un mecanismo bidding, al contratista se le permite particionarse en una tarea y premiar los contratos de otras computadoras con el propósito de mejorar el desempeño. En este caso el contratista llega a ser un manejador.

Existe un número importante de características de los algoritmos de balanceo de carga basados sobre bidding. La primera es que la efectividad y desempeño de ese algoritmo depende de la cantidad y tipo de información intercambiada durante la negociación. La segunda es que todas las computadoras tienen una total autonomía; los manejadores pueden decidir a qué computadoras premiar y los contratistas tienen la libertad de responder los mensajes enviando o no bids y si se envían a qué manejador se los envían. Algoritmos cooperativos que utilizan el enfoque bidding son discutidos por Casavant-Kuhl 1984, Ramamritham-Stankovic 1984 y Stankovic 1985 en [Goscinski 94].

2.5.3 Estrategias Emisoras Vs. Receptoras.

Un problema básico del balanceo de carga sería, ¿cuándo la migración de un proceso debe ser ejecutada?. Wang-Morris 1985 [Singhal 94] proponen una taxonomía para los algoritmos de balanceo de carga que es basada sobre el tipo de nodo en que se toma la iniciativa en la búsqueda global de computadoras descargadas y fuertemente cargadas. Indirectamente, esta taxonomía resuelve el problema de cuándo el balanceo de carga debe comenzar.

La taxonomía es basada sobre un modelo lógico en el cual los nodos son divididos en dos grupos:

emisores que generan tareas que serán procesadas y receptores que procesan esas tareas, una simple computadora pudiera ser ambos emisor o receptor, el modelo lógico es ilustrado en la Figura 2.4.

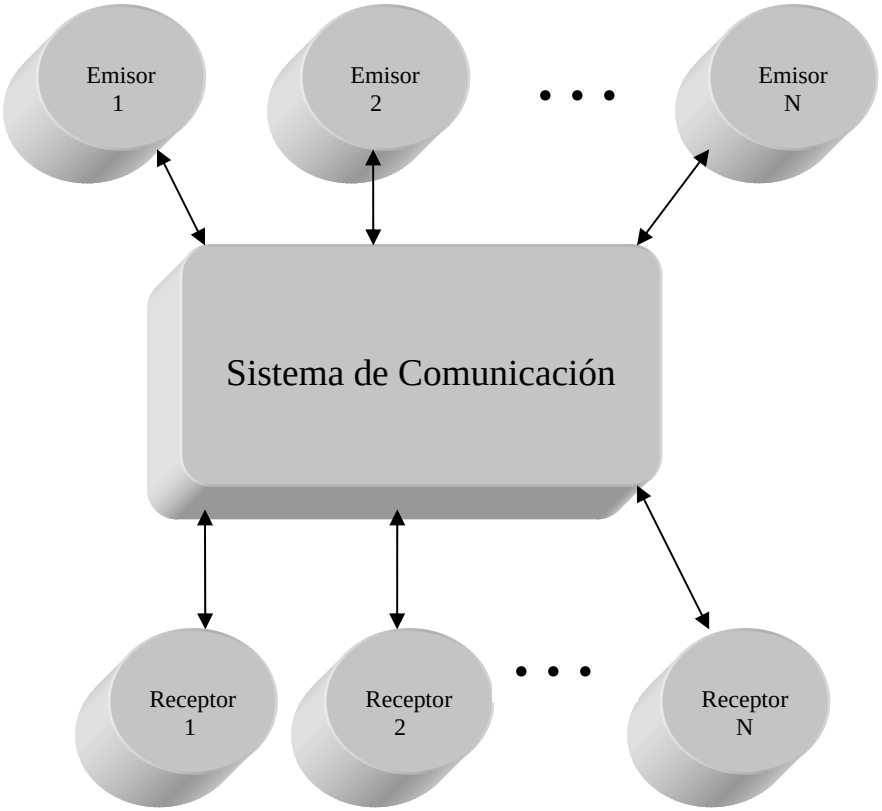


Figura 2.4 Modelo Basado en la Iniciativa para un Sistema de Computación Distribuida (Adaptado por Wang-Morris 1985 [Singhal 94]).

En un sistema distribuido el problema es qué computadoras pueden buscar computadoras descargadas. La división de los nodos anteriormente dada lleva a las estrategias emisor y receptor. Si el nodo emisor esta sobrecargado, este es responsable de obtener una localización remota para enviar un proceso, esta estrategia es llamada emisor (source-initiated), por otra parte si el servidor busca un proceso de una computadora sobrecargada la estrategia es llamada receptor (server-initiative). Los algoritmos emisores deben usarse en sistemas con cargas ligeras a moderadas y los algoritmos receptores deben usarse en sistemas con cargas altas, pero solamente si el costo de migrar un proceso es no significativo [Singhal 94]. Eager y sus colaboradores [Eager 86] notaron que los algoritmos receptores son los que mejor soportan la migración de procesos.

2.5.4 Clasificación Basada Sobre el Nivel de Dependencia de Información.

Esta clasificación es basada sobre el nivel de dependencia de información que está envuelta en una estrategia para seleccionar un nodo remoto. El nivel de dependencia de información que está envuelta en una estrategia significa el grado en que un nodo emisor conoce el estatus del servidor o un servidor conoce el estatus del emisor. Existen siete niveles de los algoritmos de balanceo de carga, que son clasificados por la dependencia de información, estos han sido identificados por Wang-Morris [Singhal 94]. Este número de niveles es arbitrario. Los niveles de información han sido organizados de tal forma que la información de más alto nivel se sobrepone al nivel más bajo. El nivel más bajo es representado por algoritmos ciegos 'blind' que no requieren información alguna. El nivel alto corresponde a esos algoritmos que requieren gran cantidad de información.

Uno puede esperar que un incremento en el nivel de información permite la construcción de mejores algoritmos, y estos mejoran el desempeño del sistema total. Esto es cierto sólo en algunos casos, por que para algún estado dado:

1) Los componentes del planeador que intercambian más información incrementan los costos de comunicación, sobrecargando las líneas de comunicación, y
2) Un software más sofisticado y complejo tiene que ser necesario, generando altas sobrecargas de computación.

El problema puede ser declarado diferente, preguntado ¿Qué nivel de complejidad es apropiado para un algoritmo de balanceo de carga dinámico? En este contexto las siguientes 3 políticas de localización pueden ser estudiadas [Singhal 94].

1. Aleatoria (Random). No usa información acerca del estado de otras computadoras un proceso es enviado a otra computadora aleatoriamente. Es decir esta política de información no intercambia información de estado. La desventaja de esta política es, que fácilmente se envía un proceso a una computadora inapropiada.
2. Umbral (Threshold). Esta colecciona y usa una pequeña cantidad de información acerca de la computadora destino. De acuerdo a esta política una computadora es seleccionada de manera aleatoria, y se evalúa la carga para verificar que no exceda un umbral. Sí la política

no excede el umbral el proceso es migrado, en otro caso la política es repetida.

3. Cola Más Corta (Shortest). Este algoritmo colecciona y usa más información que el algoritmo previo y hace la mejor elección. En este caso un número de computadoras es seleccionada en forma aleatoria y cada una es evaluada para identificar la longitud de la cola, el proceso es migrado a la computadora que tenga la longitud de cola más corta.

De acuerdo a Eager 1986 y Ramamritham 1989 [Singhal 94] la experiencia con estos algoritmos muestran que:

1) Políticas de planeación extremadamente simples, esto es, políticas que coleccionan una muy pequeña cantidad de información de estado y usan esta información de una forma muy simple, proveen un mejoramiento de desempeño dramático relativo a ningún balanceo de carga.
2) El desempeño logrado con estas políticas simples es cercana a lo que puede esperarse de una política compleja. Las políticas complejas son las que coleccionan una gran cantidad de información e intentan hacer la mejor elección dada esta información.
3) Un balanceo de carga efectivo puede ser logrado sin excesiva información de estado. Además la información de estado excesiva puede dañar el desempeño del sistema (incremento de los costos de comunicación y costos de computación).

2.5.5 Pronóstico y Futuro en el Balanceo Dinámico de Carga.

Hasta ahora los investigadores y desarrolladores en el área de balanceo dinámico de carga se han centrado sobre la identificación y evaluación de políticas eficientes para distribuir información, transferir trabajos y localizar computadoras. En el futuro un mayor intercambio de información entre programadores, compiladores y sistemas operativos será necesario. En particular, sentimos que el énfasis estará sobre el desarrollo de políticas eficientes para distribuir la información de carga y las políticas que hacen decisiones de colocación, por que esas dos áreas causan la mayor sobrecarga (overhead) del balanceo dinámico de carga. El balanceo dinámico de carga incurre en mucha sobrecarga, debido a las operaciones de transferencias de tareas, una transferencia de tareas, no tomará lugar a menos que los beneficios de relocalización sean mejores que la sobrecarga. Futuras investigaciones y desarrollos en esta área enfatizarán: [Shirazi 95]

- Planeación Híbrida (Estática/Dinámica).
- Medidas Efectivas de Índices de Carga.
- Organizaciones Jerárquicas de los Sistemas con Políticas de Información de Carga Local y Políticas de Balanceo de Carga Local.
- La Incorporación de un Conjunto de Herramientas a Nivel Sistema Operativo Distribuido, Usadas para Implementar Diferentes Políticas de Balanceo de Carga Dependiendo de la Arquitectura del Sistema y de Los Requerimientos de Aplicación.

2.6 Transferencias con Estado de Ejecución (Preemptives) Vs. Transferencias sin Estado de Ejecución (No-Preemptives).

Los sistemas de balanceo dinámicos de carga, son "sin estado de ejecución" si asignan sólo procesos recién creados a las computadoras, esto es, procesos que no son re-asignados una vez que comienzan a ejecutarse Barak and Silo 1985, Chen 1987 [Singhal 94]. Esta clase de sistemas es también llamado no migratorio Eager 1988 [Singhal 94]. En los sistemas de balanceo de carga con estado de ejecución también llamados sistemas con migración, los procesos en ejecución deben ser interrumpidos, movidos a otras computadoras y reanudados en un nuevo ambiente de computación. La migración de procesos con estado de ejecución es complicada. Pero ¿existen beneficios de desempeño para el balanceo de carga con estado de ejecución mas haya de los ofrecidos por el balanceo de carga sin estado de ejecución?. Este problema fue estudiado por Eager 1988, como resultado de su estudio, él presentó las siguientes conclusiones: [Singhal 94]

1) El balanceo de carga migratorio puede ofrecer un modesto desempeño computacional solamente en condiciones extremas. Esas condiciones son caracterizadas por la alta variabilidad en la demanda del servicio y la generación de carga de trabajo.
2) Los beneficios de las políticas con estado de ejecución no son limitadas por su costo, si no más bien por la efectividad de las políticas de balanceo no migratorias.

2.7 Metas en la Distribución de Carga.

Ahora se ilustrarán varias metas en la distribución de carga que ayudan a tener una mejor comprensión de los algoritmos de balanceo de carga. Las metas son descritas de acuerdo Mukesh Singhal y Niranjan G. Shivaratri [Singhal 94], A. Goscinski [Goscinski 94] y Hisao Kameda [Kameda 97].

2.7.1 Índices de Carga.

Zhou 1987 [Singhal 94] mostró que la longitud de cola del nodo y particularmente la longitud de cola del procesador son buenos indicadores de carga por que ellos se correlacionan bien con el tiempo de respuesta de las tareas. Medir la longitud de cola del nodo es simple y genera poca sobrecarga (overhead). Sin embargo si la transferencia de una tarea involucra un retardo significativo, el usar solamente la longitud de cola del procesador como indicador de carga puede resultar que un nodo acepte tareas mientras otras tareas que este nodo aceptó anteriormente estén aún en tránsito. Como resultado, cuando todas las tareas que el nodo aceptó han arribado, el nodo puede llegar a sobrecargarse y requiere posterior transferencia de tareas para reducir su carga. Esta situación indeseable puede ser prevenida incrementando la longitud de cola del procesador cuando el nodo acepte una tarea remota. Para evitar anomalías, cuando la transferencia de la tarea falla, un tiempo para aceptación puede ser empleado (timeout). Después de que el tiempo de aceptación termine, si la tarea aún no ha llegado, la longitud de cola del procesador es decrementada.

2.7.2 Desempeño.

Goscinski, dice que la manera de resolver un problema de balanceo de carga es enfocar el problema como un problema de optimización. En general existen 3 índices de desempeño para la planificación distribuida [Goscinski 94]:

1) Rendimiento del sistema. Este índice esta orientado principalmente hacia el desempeño computacional y mide cuanta información puede procesar en un período específico de tiempo.
2) Tiempo de espera. Definido como la cantidad total de tiempo que un proceso

consume esperando por recursos. Este índice refleja las expectativas de desempeño de los usuarios, junto con los que están orientados al desempeño de la computadora.

3) Tiempo de respuesta. El tiempo de respuesta en los sistemas distribuidos es un índice orientado hacia las expectativas de desempeño del usuario. Un índice de desempeño claramente asociado con el tiempo de respuesta es el tiempo promedio de ejecución (usado por Barak-Silo 1985 y Chen 1987 [Goscinski 94]).

2.7.3 Características de los Algoritmos de Balanceo de Carga.

Los algoritmos de balanceo de carga dinámicos deberían desarrollarse con el respaldo de las siguientes 5 características, que han tenido un fuerte impacto sobre el desempeño Fogg 1987 : [Goscinski 94]

1) Estimación de carga. La carga de la computadora involucra procesos que serán ejecutados y los que están en ejecución. La estimación de carga debe incluir procesadores, procesos y características del ambiente.
2) Intercambio de información entre computadoras. Una computadora transmite la información de carga para informar a computadoras remotas de su estado actual. La cantidad de información intercambiada involucra un convenio (tradeoff) entre tener un alto nivel de detalle (por ejemplo, información acerca de todos los procesos) y minimizar el volumen de tráfico de la red.
3) Información de carga y patrones de transmisión. Una computadora debe enviar información de carga a todas las computadoras remotas, cuando quiere migrar procesos. Dos patrones de direccionamiento deben de ser usados: uno a uno y uno a muchos. El problema de que tan a menudo estimar la carga es aun un problema abierto.
4) Decisiones para mover tareas. Un mecanismo de decisión debería ser usado para determinar si un proceso particular se ejecuta bien o no en una computadora dada.
5) El receptor debe conocer las tareas que van a migrar. Es preferible que una computadora destino conozca que procesos arribarán antes de que ellos lleguen. Procesar esta información ayuda a evitar algunos de los problemas de información desactualizada.

2.7.4 Estabilidad.

Se describen dos vistas de estabilidad.

1. Desde el punto de vista de teoría de colas. Cuando la razón de arribo es mayor que la razón de procesamiento del sistema, las colas crecen sin límite y el sistema es declarado inestable. Por ejemplo, considere un algoritmo de distribución de carga que realiza excesivo intercambio de mensajes para coleccionar información de estado. La suma de la carga debido al trabajo de arribo externo y la carga debido al intercambio de mensajes impuesto por el algoritmo puede llegar a ser mayor que la capacidad de servicio del sistema, causando inestabilidad en el sistema.
2. Desde el punto de vista del algoritmo. Si un algoritmo puede ejecutar acciones infructuosas indefinidamente con una probabilidad finita, el algoritmo se dice que es inestable. Por ejemplo, considere el procesamiento de basura (processor thrashing), la transferencia de una tarea a un receptor pudiera incrementar la longitud de cola del receptor hasta el punto de sobrecargarlo, necesitando la transferencia de la tarea a otro nodo. Este proceso pudiera repetirse indefinidamente. En este caso, una tarea es movida de un nodo a otro en busca de un nodo ligeramente cargado sin recibir servicio, el sistema eventualmente entrará en un estado en que los nodos están gastando todo su tiempo en transferir las tareas y no en ejecutarlas, causando que el sistema llegue a un estado sumamente inestable.

2.8. Conclusiones.

Un algoritmo de balanceo dinámico de carga consiste de tres políticas principales: una política de información, una política de transferencia y una política de localización, que determina a qué nodo una tarea seleccionada deberá de ser enviada. Los algoritmos de balanceo de carga pueden ser clasificados usando varias características. Hay dos clasificaciones básicas: jerárquica y compacta (flan), acorde a esta clasificación un algoritmo cooperativo, optimo o sub-óptimo, etc. puede usarse para balancear la carga cuando queremos mejorar el desempeño de un sistema de computación distribuida. Si las computadoras cooperan y no tienen exactitud de los datos, ellas pudieran usar un algoritmo con enfoque bidding (ofrecimiento) para seleccionar una computadora remota. Estrategias emisoras o

receptoras pueden usarse para determinar que computadora puede iniciar el balanceo de carga. La estimación del estado del sistema actual y la toma de decisiones son dos componentes en la planificación dinámica de tareas, usar buenos indicadores de carga que correlacionen bien estos dos componentes contribuiría en el mejoramiento del tiempo de respuesta global de una aplicación distribuida.

Capítulo 3. Algoritmos para Balancear Carga Dinámicos.

Este capítulo tiene como objetivo ilustrar diferentes algoritmos de balanceo de carga dinámicos que han aparecido en la literatura y que caen en la taxonomía mencionada en el capítulo 2.

Andrzej Goscinski selecciona cinco algoritmos que corresponden a la clasificación jerárquica del apartado 2.3 [Goscinski 94].

1. Algoritmo de Bryant y Finkel´s (también llamado algoritmo de paridad). Pares de procesadores intercambian procesos.

2. Algoritmo de Barak y Shiloh´s (también llamado algoritmo del vector). Los procesadores mantienen vectores de carga.

3. Algoritmo de Stankovic y Sidhu´s (también llamado algoritmo bidding). Las computadoras cooperan usando un enfoque bidding.

4. Algoritmo SLA que es basado sobre un autómata de aprendizaje estocástico.

5. Algoritmo BDT que es basado sobre teoría de decisiones bayesianas.

Eager, Lazowska y Zohorjan cubren tres algoritmos simples pero aún efectivos que caen en el apartado 2.6. Estos algoritmos son emisores, en esta clase de algoritmos la actividad de distribución de carga es iniciada por un nodo sobrecargado (emisor) que intenta enviar una tarea a un nodo descargado (receptor). Los algoritmos son Aleatorio, Cola Más Corta y Umbral [Eager 86]. Este capítulo concluye mencionado algunos algoritmos y políticas que caen en el apartado 2.6.1 y 2.6.3 del capítulo 2. Los algoritmos son el simétrico (Above-Average) propuesto por Krueger y Finkel [Singhal 94] y dos políticas con un enfoque adaptativo propuestas por Krueger y Shivaratri [Krueger 94].

3.1 Algoritmo de Bryant y Finkel´s.

El algoritmo de Bryant y Finkel´s es un algoritmo dinámico y físicamente distribuido. Para tomar una decisión, las computadoras cooperan enviando la negociación de mensajes. Las decisiones son sub-optimas y un enfoque heurístico es usado para obtener una solución. Este algoritmo distribuido mejora el desempeño localmente. Este algoritmo ha sido escogido por que es un buen ejemplo de un algoritmo basado en la cooperación.

Estratégicamente trabaja de la siguiente forma:

1) Una computadora C_A envía una pregunta a uno de sus vecinos más cercanos C_B para formar un par temporal, un ambiente estable y controlado esta situado para la migración del trabajo. La pregunta tiene dos propuestas:

- Se informa a la computadora C_B que C_A desea formar un par, y
- Que esta contiene una lista de procesos y el tiempo consumido por cada proceso en C_A. El tiempo consumido por cada proceso es usado por un procedimiento que estima la carga, para estimar el tiempo restante de los requerimientos del proceso.

2) C_B, después de recibir la pregunta, puede llevar a cabo una de las tres opciones:

- Rechaza las preguntas $C_A´s$, esto implica que C_A debe enviar una pregunta a otro vecino cercano.
- Formar un par con C_A; esto implica que C_A también como C_B rechazan todas las preguntas que lleguen hasta que el par se haya roto.
- Posponer C_A cuando C_B está en un estado de migración, es decir, enviando procesos, esto implica que C_A debe esperar hasta que C_B forme una par con este, o rechace este, C_A no puede preguntar a nadie más.

3) Después de establecer un par, la computadora con mayor carga (por ejemplo, C_A) procede a seleccionar un proceso ($i, i = 1,...P_A$, donde P_A es un número de procesos sobre C_A) para migrar a otra

computadora. Esta operación puede hacerse sobre las bases del mejoramiento de tiempo de respuesta por cada proceso, denotado por k_i, usando la siguiente fórmula:

$$K_i = T_{Ai} / (T_{Bi} + T_{ABi}) \tag{3.1}$$

Donde:

T_{Ai}. Es el tiempo de respuesta esperado para el proceso i cuando este corre sobre una computadora C_A.

T_{Bi}. Es el tiempo de respuesta esperado para el proceso i cuando éste esta corriendo sobre una computadora destino C_B.

T_{ABi}. Es el tiempo de transferencia para el proceso i de una computadora C_A a una computadora C_B.

El tiempo de respuesta esperado de un proceso $T_{Ai}, i = 1,..., P_A$, puede ser obtenido usando la estimación de tiempo restante del requerimiento $T_E(i)$; el tiempo para completar un proceso es igual al tiempo usado a lo largo de T; y es calculado sobre las bases del siguiente algoritmo:

$T = T_E(i)$;
for all $j, j = 1,...,P_A$ **do**
begin
    if $T_E(j) < T_E(i)$
        then $T = T + T_E(j)$
    else $T = T + T_{E(i)}$
end;
$T_{Ai} = T$

4) Si ningún proceso puede esperar mejor desempeño sobre una computadora C_B, esto es, cuando $k_i < 1$, para toda $i, i = 1,..., P_A$, entonces la computadora C_A informa a C_B de este hecho y el par es roto. En otro caso, el proceso que puede esperar el mejor desempeño es seleccionado y migrado. Este

procedimiento es repetido para todos los procesos restantes, hasta que ningún proceso pueda esperar un mejor desempeño sobre una computadora C_B.

Este algoritmo no puede ser clasificado como un algoritmo emisor (source-initiative) o receptor (server-initiative). Cualquier computadora en un par, puede seleccionar un proceso para ser migrado y éste inicia la migración.

3.2 Algoritmo de Barak y Shiloh´s.

El algoritmo de Barak y Shiloh´s es un algoritmo de balanceo de carga dinámico global. Además es distribuido en el sentido de que cada computadora usa la misma política. Este algoritmo no usa conocimiento a priori para estimar los recursos que requiere cada proceso. Cada computadora mantiene información de la carga de otras computadoras. Esta información es frecuentemente intercambiada usando un camino de selección aleatoria. Este algoritmo corresponde a la clase de algoritmos cooperativos. Este algoritmo es muy simple, el balanceo de carga esta enfocado a reducir la varianza de carga entre computadoras de un sistema distribuido. El algoritmo de Barak y Shiloh´s es un algoritmo receptor (server-initiative).

Barak y Shiloh´s asumen que:

1. N es el número de computadoras en el sistema y N es un número muy grande.
2. Existe alguna comunicación directa que se liga entre un par de nodos.

La política de balanceo de carga consiste en tres componentes distintos: un algoritmo de carga del procesador, un algoritmo de intercambio de información y un algoritmo de migración de procesos. Solamente se analizarán los dos primeros, que son algoritmos orientados a políticas:

El algoritmo de carga del procesador es usado por cada computadora para monitorear y estimar su carga local. La carga es calculada por cada unidad en un tiempo q, y es promediada en un periodo de tiempo t. Esto es, $L(0)$ es la carga estimada de una computadora local y su valor es actualizado

periódicamente. Esta información forma el primer componente de un vector de carga L, que tiene varios vectores pequeños de tamaño (l). El resto de los componentes toman los valores de un subconjunto arbitrario de procesadores.

El algoritmo de intercambio de información es responsable del intercambio continuo de la información de carga (vectores de carga) entre computadoras; $L(j)$ denota el componente j^{th} del vector de carga, $0 \le j \le l-1$, que es la carga estimada de la j^{th} computadora. Porciones del vector de carga L es frecuentemente intercambiado usando un camino de selección aleatoria, de una forma similar a un programa worm. Tal intercambio proporciona información suficiente y actualizada de cada computadora.

El algoritmo ha sido construido en tal forma que la información de carga de los trabajos recién creados sea colocada en posiciones que garanticen que los últimos valores de carga sean mantenidos. El algoritmo de intercambio de información trabaja en la siguiente forma. Cada unidad de tiempo sobre cada computadora.

1. Actualiza su propio valor de carga.
2. Escoge aleatoriamente otra computadora i, $1 \le i \le N$,
3. Envía la primera mitad del vector de carga local L, a la computadora i.

Después de recibir una porción del vector de carga, cada computadora:

- Usa los vectores recibidos para estimar la carga de cada computadora. Para hacer esto, el receptor primero mezcla cada vector recibido con su propio vector de carga, L, de acuerdo al siguiente mapeo:

$$L(2i) = L(i) \qquad 1 \le i \le l/2 - 1$$

y

$$L(2i+1) = L_r(i) \quad 0 \le i \le l/2 - 1 \tag{3.2}$$

- Estima la carga de una computadora para obtener el número promedio de procesos solicitando un servicio en cada periodo de tiempo. El resultado puede ser usado para estimar el tiempo de respuesta de ese procesador.

- Migra un proceso a una computadora que provee el tiempo de respuesta estimado más bajo. Un proceso especial en cada computadora periódicamente causa que todos los otros procesos consideren la migración.

El alcance del mejoramiento del desempeño depende del valor de l. Si l es igual al número de nodos en la red, N, este algoritmo es globalmente óptimo.

Este algoritmo no trabaja eficientemente cuando un gran número de procesos es repentinamente creado. En este caso el límite del algoritmo sobre el número de procesos migrándose implica excesiva carga computacional.

3.3 Algoritmo de Stankovic y Sidhu´s.

El algoritmo de Stankovic y Sidhu´s también corresponde a la clase de algoritmos globales, dinámicos y físicamente distribuidos. Este algoritmo requiere cooperación entre computadoras, utilizando un enfoque bidding para hacer una decisión. Las decisiones hechas son sub-óptimas y heurísticas. Ellas son basadas sobre la historia de la actividad del sistema. Este algoritmo esta incluido aquí, por que es un claro ejemplo de un algoritmo adaptativo y bidding. Note que el algoritmo de Stankovic y Sidhu´s es un algoritmo emisor (source-initiated).

La heurística utiliza información a priori concerniente a las características de los procesos. Esas características son requerimientos de recursos, necesidades de recursos especiales, prioridad de los procesos, restricciones de precedencia y las necesidades de grupos distribuidos y agrupamientos. El algoritmo utiliza el procedimiento de evaluación McCulloch-Pitts (MPEP). Este algoritmo es basado sobre una célula de decisión que tiene un número de entradas exitatorias e inhibitorias y un simple valor de salida. La salida es la suma de las exitatorias o cero si alguna de las inhibitorias esta colocada.

El algoritmo trabaja de la siguiente forma:

1. Todos los procesos locales en un host, son periódicamente evaluados utilizando un MPEP para decidir si se transmite un request-for-bid para un proceso en particular. La entrada al MPEP incluye características de la red y del proceso.

2. Si la salida del MPEP esta sobre algún umbral (threshold), entonces el proceso se esta ejecutando bien en el ambiente actual. En otro caso, si la salida no es cero, el proceso debe de ser movido, por que se esta ejecutando pobremente. Si el MPEP retorna cero, como resultado de colocar alguna de las células inhibitorias, el proceso debe de no ser movido.

3. Cuando al menos algún proceso en la computadora local necesita ser movido, la computadora local realiza un broadcast llamado request-for-bit (RFB) a todas las otras computadoras con una distancia i, $i = 1,...,n$, donde n, es el diámetro de la red (tratado como un parámetro del algoritmo).

   - El mensaje RFB es enviado por cada proceso que necesita ser movido.

   - El RFB contiene todas las características de migración. Es decir todos los receptores, pudieran correr el MPEP para determinar si el proceso pudiera correr bien (esto es, el valor del MPEP debería estar sobre el umbral) si este fuera migrado.

4. Después de algún periodo de tiempo predefinido t, todas las respuestas al mensaje RFB son ajustadas por el costo de transferir el proceso. El mejor bid es entonces considerado como la computadora destino. Una evaluación adicional es entonces ejecutada antes de que la acción sea tomada:

   - El valor del MPEP para el proceso es evaluado. Si el proceso esta aún ejecutándose pobremente, entonces el proceso se migra al bidder ganador, en otro caso este se retiene en la computadora actual.

5. Si durante un periodo de tiempo t, ningún bid es recibido, la distancia i, para enviar el RFB es incrementada, y todo el algoritmo es repetido.

Es necesario enfatizar dos hechos. Primero, la parte adaptativa de este algoritmo es que dinámicamente modifica el número de saltos que el RFB admitirá, dependiendo de las condiciones actuales. Segundo, el algoritmo de balanceo de carga mejora el desempeño con cierto alcance dependiendo de n. Si n es igual al número de nodos en la red, este algoritmo es globalmente óptimo.

3.4 Algoritmo SLA, Basado Sobre un Autómata de Aprendizaje Estocástico.

Este algoritmo corresponde a la clase de algoritmos globales, dinámicos y físicamente distribuidos. Las computadoras cooperan para tomar una decisión. Las decisiones son basadas sobre un enfoque heurístico. El algoritmo adapta sus parámetros, basados en la historia del comportamiento del sistema. Utiliza una red para un autómata de aprendizaje estocástico que hace uso de la retroalimentación de información que identifica si una decisión de balanceo de carga previa fue benéfica o no. Este algoritmo contiene un mecanismo explícito para tratar con la estabilidad. Este corresponde a la clase de algoritmos receptores (server-receive). Cada computadora de un sistema distribuido:

- Contiene un algoritmo de balanceo de carga, cada algoritmo es modelado como un SLA.

- Tiene una carga óptima (la que espera en la cola, mas la que esta en ejecución). Si una computadora tiene menos que un número de procesos en ejecución o en la cola de espera, esta computadora esta descargada, si tiene más, esta sobrecargada. Esta es una extensión al concepto original de autómata de aprendizaje estocástico [Goscinski 94].

En el desempeño de balanceo de carga debe ser posible que cada computadora reconozca computadoras descargadas, esto es, debe de ser posible identificar estados de la red.

- Si hay N computadoras, se podrían tener 2^N posibles estados. Este es el caso peor.

- Para proveer un rápido balanceo en la red y direccionar la estabilidad, el algoritmo usa una

heurística que reduce el número de estados para reconocer un número limitado de computadoras descargadas en un tiempo. Esos estados son llamados, estados de red.

- Un SLA básico tiene un vector de probabilidad con el cual este selecciona posibles acciones. Este modelo ha sido extendido agregando el concepto de estados de red, por lo tanto hay un vector de probabilidad por cada estado de la red. Esto es, los vectores de acciones y probabilidades de un SLA básico ahora llegan a ser una matriz, con un renglón por cada estado de red. Este es también una extensión al concepto original de autómata de aprendizaje estocástico y es llamado estados de red por naturaleza.

Las operaciones del algoritmo de planeación son como siguen:

1. Cada computadora periódicamente, con un periodo $T1$, checa su propio estatus (descargado o sobrecargado) y realiza un broadcasts de este estatus a todas las otras computadoras.

2. Esta información es usada por cada computadora para determinar el estado de red que una computadora en particular está observando en ese tiempo. Este determina el vector de probabilidad a usar y el conjunto de acciones asociadas con este vector para usarse en la transmisión de tareas en ese periodo de tiempo.

3. En paralelo, el algoritmo de balanceo de carga verifica, en un periodo $T2$, si la computadora esta sobrecargada. Si esta, ésta sólo transmitirá un proceso a otra computadora, que es escogida probabilísticamente (sobre las bases del vector de probabilidad que esta actualmente en efecto).

4. Cuando la computadora destino recibe la migración del proceso, ésta decide si respaldar o penalizar la acción basada en el efecto de arribo del nuevo proceso sobre su propia carga.

5. La decisión (respaldada o penalizada) es transmitida de regreso a la computadora emisora que actualiza el vector de probabilidad para tomar en cuenta el efecto de la acción (la probabilidad es decrementada si la acción es penalizada, o incrementada cuando la acción fue respaldada).

El mejoramiento de desempeño de este algoritmo depende de la heurística en uso.

3.5 Algoritmo Basado Sobre la Teoría de Decisiones Bayesianas.

El algoritmo de Stankovic´s es un algoritmo global y dinámico que requiere cooperación entre computadoras para tomar una decisión. Las soluciones son sub-óptimas y son logradas utilizando un enfoque heurístico basado en la teoría de decisiones Bayesianas. Este es un algoritmo de balanceo de carga físicamente distribuido soportado por una unidad de monitor central. La tarea del monitor es coleccionar e informar a las computadoras acerca de los cambios en su ambiente. Esto es, no usa un enfoque totalmente distribuido, sin embargo se considera por que tiene dos características únicas e interesantes. La primera es que la pérdida de la unidad de control no afectara el desempeño de todo el algoritmo. La segunda esta en la arquitectura: la función de asignación de carga es dividida en dos partes: un planeador de trabajos descentralizado (DJS) y un planeador de procesos descentralizado (DPS).

El papel del DJS, que esta compuesto de múltiples entidades y distribuidas es como sigue. Asumiendo que cada computadora mantiene una cola de espera para los trabajos que arriban:

El DJS intenta mantener un número de trabajos esperando en cada computadora en forma ecualizada, de tal forma que se mejore el tiempo de respuesta y se realice esto con bajo costo de ejecución. Esto es logrado a través de una reasignación de trabajos. Este mantiene información de estado acerca de la red, de tal forma que continuamente se adapte al estado de la red. Sin embargo el DJS no conoce acerca de las características de los trabajos que llegan.

Para disminuir la sobrecarga (overhead) del DJS, no se ejecuta frecuentemente. Esto es posible, ya que las computadoras utilizadas por el monitor y las heurísticas no son muy caras. En la práctica, solamente un monitor trabaja constantemente. El segundo monitor es usado solamente cuando el primero falla. La adaptación heurística esta basada sobre la teoría de decisiones Bayesianas. Esto es, esos monitores están disponibles para tratar con la dinámica del sistema.

El DPS decide cual de los trabajos que están esperando, serán re-activados. El DPS trata con múltiples

módulos de un trabajo, sus requerimientos de recursos, agrupamientos y asignación, durante la ejecución de un trabajo (proceso). Los trabajos (en el caso del DJS) y módulos (en el caso del DPS) pudieran migrar algún número de veces, sobre algún limite predefinido. El articulo de Stankovic solamente trata con el uso de teoría de decisiones Bayesianas como base para el DJS. El DPS no es discutido.

Este algoritmo de la teoría de decisiones Bayesianas es en muchas formas similar a un SLA. La más importante diferencia es que no hay inmediata retroalimentación de otras computadoras que estén atadas a la red. Además una distinción se hace entre el estado de la red y la computadora que observa este estado.

La operación del algoritmo de balanceo de carga es como sigue:

1. Cada computadora envía (en un periodo P) al monitor:

    - La observación del estado de la red, y
    - El estatus, que es, el número de procesos locales.

2. Sobre las bases de esta información, el monitor calcula:

    - El estado exacto de la red, y
    - Por cada computadora, el estado observado.

    Esto significa que para alguna observación, el monitor calcula el estado de la red más probable.

3. El monitor transmite (en un periodo S) este vector a cada computadora.

4. Cada computadora hace un broadcasts (en un período T) para estimar la carga de las demás computadoras, por lo tanto cada computadora puede hacer una observación del estado de la red.

5. Cuando la computadora ha hecho una observación, una acción es ejecutada: la acción debe ser, por ejemplo, no hacer nada, migrar un proceso a una computadora A. El tipo de acción es determinada por una función de utilidad estática en la cual la acción con alta utilidad es preferida.

Stankovic mostró que los parámetros P y T pueden ser afinados para lograr un desempeño óptimo en determinado costo [Goscinski 94].

3.6 Algoritmos Emisores (Sender-Initiated).

En los algoritmos emisores, la actividad de distribución de carga es iniciada por un nodo sobrecargado (emisor) que intenta enviar una tarea a un nodo descargado (receptor). Esta sección cubre tres algoritmos simples pero aún efectivos, estudiados por Eager, Lazowska y Zohorjan [Eager 86]. Los algoritmos son el Aleatorio, Cola Más Corta y Umbral.

Política de Transferencia. Los tres algoritmos utilizan la misma política de transferencia, una política umbral (threshold) basada en la longitud de cola del CPU. Un nodo es identificado como un emisor, si una tarea que ha surgido en el nodo hace que la longitud de la cola exceda el umbral T. Un nodo se identifica así mismo como un receptor para una tarea remota, si aceptando la tarea no causara que la longitud de cola exceda el umbral.

Política de Selección. Los algoritmos emisores aceptan tareas recién creadas para ser transferidas.

Política de Localización. Los algoritmos difieren solamente en su política de localización.

3.6.1 Algoritmo Aleatorio (Random).

Este algoritmo utiliza una simple política de localización dinámica que no utiliza la información de estado remota. Una tarea es transferida a un nodo seleccionado en forma aleatoria sin ningún intercambio de información entre nodos. Un problema con este enfoque es que una transferencia de tarea sin ningún uso (useless) puede ocurrir cuando una tarea es transferida a un nodo que esta

fuertemente cargado [la longitud de la cola esta sobre el umbral (threshold)]. Una meta que surge con esta política concierne a la siguiente pregunta: ¿Cómo un nodo pudiera tratar una tarea transferida?. Si ésta es tratada como un nuevo arribo, la tarea puede ser transferida a otro nodo, si la longitud de cola local esta sobre el umbral. Eager [Eager 86] ha mostrado que si éste es el caso, sin considerar la carga promedio del sistema, el sistema eventualmente entrara en un estado en que los nodos estarán gastando todo su tiempo en transferir tareas y no en ejecutarlas. Una simple solución a este problema es limitar el número de veces en que una tarea pueda ser transferida.

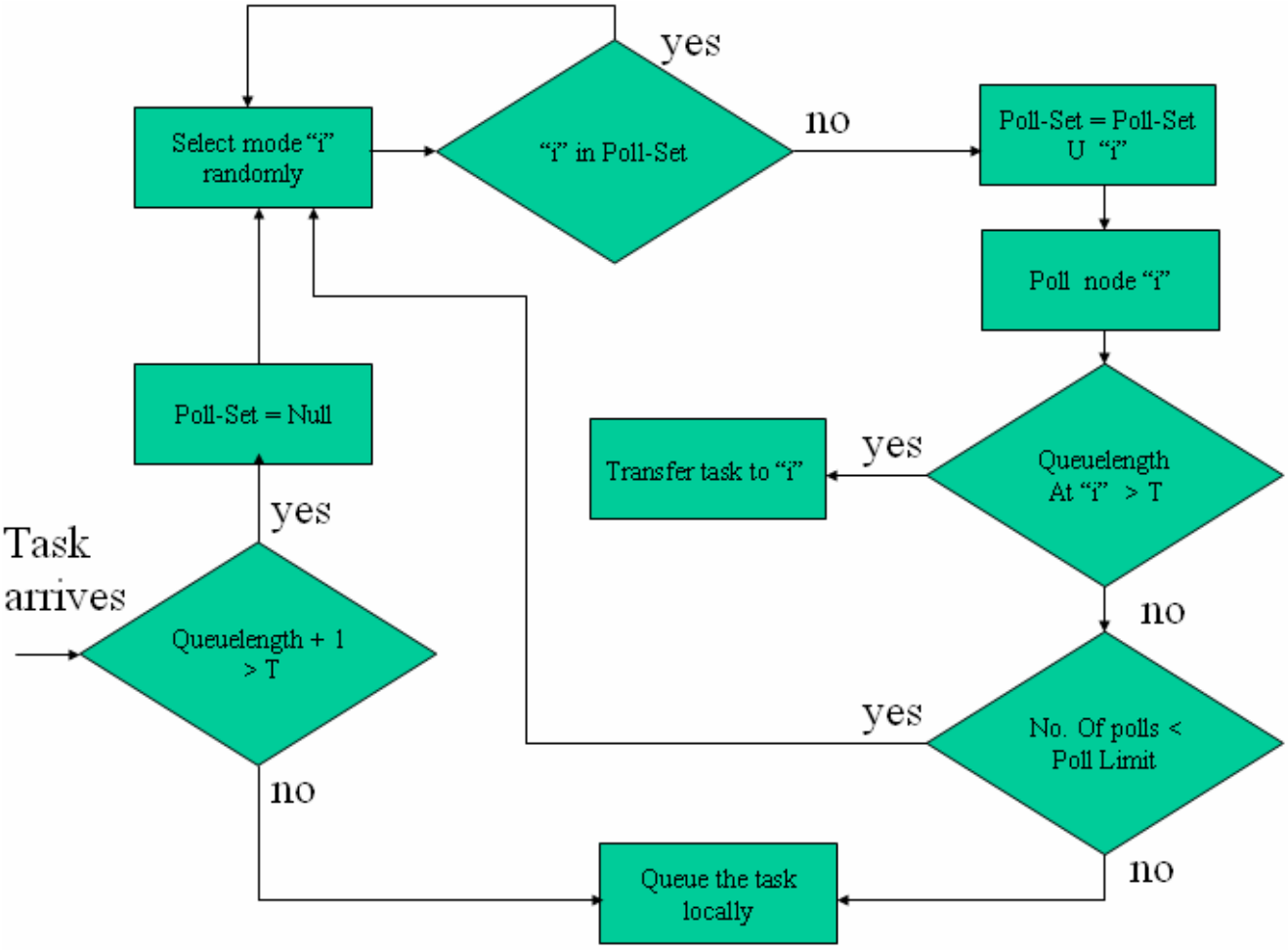


Figura 3.1 Algoritmos Emisores con Políticas de Localización Umbral (Threshold).

3.6.2 Algoritmo Umbral (Threshold).

El problema de la tarea transferida que no tendrá ningún uso en un algoritmo aleatorio (Random) puede ser evitada estimando la carga de un nodo seleccionado de forma aleatoria, para determinar si éste es un receptor (Ver Figura 3.1). Por lo tanto la tarea es transferida al nodo seleccionado, que debería ejecutar la tarea respaldando el estado cuando la tarea llegue. En otro caso, otro nodo es seleccionado de forma aleatoria y la carga es estimada nuevamente. El número de estimaciones esta limitado por un parámetro llamado PollLimit, con el propósito de mantener la sobrecarga (overhead) baja. Si ningún nodo receptor es obtenido en el PollLimit, entonces el nodo donde se origina la tarea deberá ejecutarla. Para evitar la transferencia de tareas sin uso, la política umbral provee un desempeño substancial sobre la política de localización aleatoria [Eager 86b].

3.6.3 Algoritmo de la Cola Más Corta (Shortest).

Los dos enfoques previos no enfocan la selección del mejor receptor. En el algoritmo de cola más corta un número de nodos (= PollLimit) es seleccionado en forma aleatoria y se estima la carga para determinar la longitud de la cola [Eager 86]. El nodo con la longitud de cola más corta es seleccionado como el destino para transferir la tarea a menos que la longitud de cola sea mayor o igual a T. El nodo destino ejecutara la tarea respaldando la longitud de la cola en el momento en que la tarea sea transferida. El mejoramiento de desempeño de la política de localización de cola más corta sobre la política umbral fue casi nulo [Eager 86], indicando que si utilizamos una política de información mas detallada, no necesariamente resulta un mejoramiento significativo en el desempeño del sistema.

Política de Información. Cuando la política de localización umbral o cola más corta es utilizada, la actividad de estimación de carga comienza cuando la política de transferencia identifica un nodo como emisor de una tarea, aquí la política de información puede ser considerada para ser del tipo sobre demanda (demand-driven).

Estabilidad (Stability). Los tres enfoques de políticas de localización utilizados en los algoritmos emisores causan inestabilidad en el sistema cuando las cargas son altas, ya que ningún nodo mantiene carga ligera y la probabilidad de que un emisor pudiera obtener un receptor es muy baja. La actividad

de estimación de carga se incrementa a razón que se incrementan los arribos en el sistema, alcanzando un punto donde el costo de compartir la carga es mayor que los beneficios. En este punto la mayoría de ciclos CPU´s son gastados en estimaciones de carga no exitosas y en responder a estas estimaciones, cuando la carga debido a los arribos y debido a la compartición de carga excede la capacidad del sistema, la inestabilidad ocurre.

3.7 Algoritmos Receptores (Receiver-Initiated).

En los algoritmos receptores, la actividad de distribución de carga es iniciada de un nodo descargado (receptor) que intenta obtener una tarea de un nodo sobrecargado (emisor) [Eager 86].

Política de transferencia. La política de transferencia es una política umbral (threshold), dónde las decisiones son basadas sobre la longitud de cola del CPU. La política de transferencia es seleccionada cuando una tarea parte, si la longitud de cola esta por debajo de un umbral T (threshold), el nodo es identificado como un receptor que obtiene una tarea de un emisor que será determinado de la política de localización. Un nodo es identificado para ser un emisor si la longitud de la cola excede el umbral (threshold).

Política de selección. Los algoritmos receptores consideran tareas con estado de ejecución para ser transferidas, aunque algunas también consideran la transferencia de tareas en el momento de su creación (tareas sin estado de ejecución).

Política de localización. En esta política un nodo seleccionado en forma aleatoria es estimado (polled), para determinar si el transferir una tarea de este nodo pudiera colocar la cola debajo del umbral (threshold). Si no el nodo estimado transfiere la tarea. En otro caso algún otro nodo es seleccionado en forma aleatoria y el procedimiento es repetido hasta que un nodo que pueda transferir una tarea (emisor) sea obtenido o se obtenga un límite de estimaciones (PollLimit), si todas las estimaciones fallan al intentar obtener un emisor, el nodo esperara un determinado periodo de tiempo e intentara nuevamente (Ver Figura 3.2).

Política de Información. La política de información es sobre demanda porque la actividad estimación

comienza solamente después de que el nodo ha llegado a ser un receptor de carga.

Estabilidad. Los algoritmos receptores no causan inestabilidad por la siguiente razón. En cargas altas existe una alta probabilidad de que un receptor obtenga un emisor para compartir la carga con muy pocas estimaciones. Esto resulta en un uso efectivo de estimaciones de los receptores y existe muy poco desgaste de ciclos CPU en el sistema. En cargas bajas del sistema existen muy pocos emisores, existiendo más estimaciones de los algoritmos receptores, pero estas estimaciones no causan inestabilidad en el sistema. La desventaja de los algoritmos receptores es que la mayoría de las tareas transferidas son con estado de ejecución y por lo tanto caras.

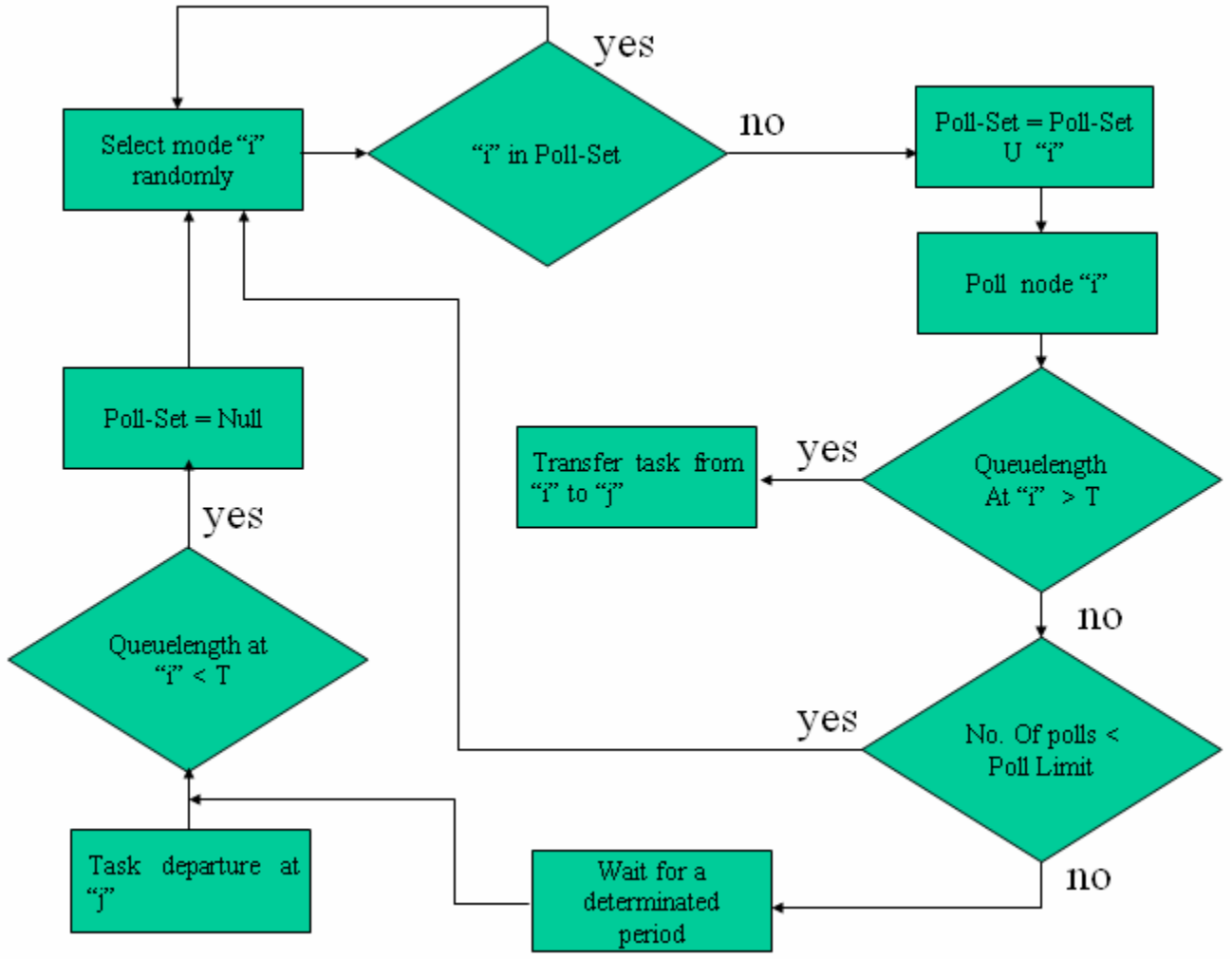


Figura 3.2 Algoritmos de Planeación de Carga Receptores (Receiver-Initiated).

3.8 Algoritmos Simétricos (Symmetrically-Initiated).

En los algoritmos iniciados simétricamente, ambos emisor y receptor buscan un emisor o un receptor para transmitir una tarea, este algoritmo tiene las ventajas del emisor y del receptor. Cuando la carga es baja el emisor es más exitoso para obtener un nodo descargado. Cuando la carga del sistema es alta, el receptor es más exitoso para obtener un nodo sobrecargado. Sin embargo estos algoritmos no son inmunes a las desventajas que ofrecen los algoritmos emisores y receptores. Un algoritmo iniciado simétricamente puede ser construido utilizando las políticas de localización y de transferencia de los dos algoritmos (emisor y receptor). Otro algoritmo simétrico es el algoritmo llamado Above-Average que es el que se describe a continuación [Singhal 94].

3.8.1 Algoritmo Above-Average.

El algoritmo Above-Average propuesto por Krueger y Finkel 1984 [Singhal 94], intenta mantener la carga de cada nodo en un rango aceptable, de tal forma que este dentro de la carga promedio del sistema. La descripción del algoritmo es como sigue:

Política de Transferencia. La política de transferencia es una política umbral que utiliza dos umbrales adaptativos. Esos umbrales son equidistantes de los nodos que estiman la carga promedio a través de todos los nodos. Por ejemplo si los nodos estiman que la carga promedio es 2, entonces el umbral bajo es 1 y el umbral alto es 3, la carga de un nodo que es menor que el umbral bajo, es considerado como un receptor, mientras que un nodo con carga mayor que el umbral alto es considerado como un emisor. Los nodos que tienen las cargas entre los dos umbrales son considerados en un rango aceptable, y no son ni emisores ni receptores.

Política de localización. La política de localización tiene los siguientes dos componentes.

Componente Emisor (Sender-Initiated).

- Un emisor (un nodo que tiene una carga mayor que el rango aceptable) realiza un broadcast con un mensaje *TooHigh*, colocando una alarma timeout al *TooHigh*, y escucha un mensaje Accept hasta que el timeout expire.
- Un receptor (un nodo que tiene la carga menor que el rango aceptable) recibe un mensaje *TooHigh*, cancela el timeout *TooLow*, envía un mensaje Accept al emisor del mensaje *TooHigh*, incrementa el valor de carga (tomando en cuenta que la tarea se ha recibido), y coloca un timeout AwaitingTask. Incrementa el valor de carga para aceptar una tarea remota. Si el timeout *AwaitingTask* expira sin el arribo de una tarea, el valor de la carga en el receptor es decrementado.
- Sobre el arribo del mensaje *Accept*, si el nodo es aún un emisor, escoge la mejor tarea para transferirse y la transfiere al nodo que respondió.
- Sobre la expiración del timeout TooHigh. Si ningún mensaje Accept ha sido recibido, el emisor infiere que la estimación de carga promedio es muy baja (dado que ningún nodo tiene una carga menor). Para corregir este problema, el emisor realiza un broadcast con un mensaje *ChangeAverage* para incrementar la carga promedio estimada en los otros nodos.

Componente Receptor.

- Un nodo que llega a ser un receptor, realiza un broadcast enviando un mensaje TooLow, coloca una alarma timeout a TooLow, y comienza escuchando un mensaje TooHigh.
- Si el mensaje TooHigh es recibido, el receptor realiza la misma acción que realiza un emisor.
- Si el timeout TooLow expira, antes de recibir un mensaje TooHigh, el receptor realiza un broadcast enviando un mensaje ChangeAverage para decrementar la carga promedio estimada en los otros nodos.

Política de Selección. Los algoritmos emisores aceptan tareas recién creadas para ser transferidas y tareas con estado de ejecución.

Política de Información. La política de información de este algoritmo es sobre demanda (demand-drive). La ventaja de este algoritmo es que la carga promedio del sistema es determinada individualmente en cada nodo, generando poca sobrecarga sin el intercambio de muchos mensajes. Otro punto clave es que los rangos aceptables determinan la responsabilidad del algoritmo. Cuando la

red de comunicación esta fuertemente cargada o ligeramente cargada (indicados por los retardos de los mensajes cortos o largos respectivamente), el rango aceptable puede ser incrementado o decrementado en cada nodo y por lo tanto las acciones de balanceo de carga se adaptan al estado de la red de comunicación también.

3.9 Algoritmos Adaptativos.

Un algoritmo simétricamente estable, agrupa la información durante la estimación de carga (en vez de descartarla como se hizo en los algoritmos previos) para clasificar los nodos del sistema.

3.9.1 Dos Políticas Adaptativas para Planeación Global.

Shivaratri y Krueger [Krueger 94] proponen dos políticas de planeación adaptativas. La primera de las políticas es una política simétrica (symmetrically-initiated) y es usada en sistemas que tienen capacidad de transferir tareas recién creadas así como tareas que mantienen una ejecución parcial. La segunda es una política emisora y es usada en sistemas donde solo los trabajos recién creados pueden ser transferidos. Esas políticas se adaptan a las características globales de la carga del sistema usando información que es normalmente descartada por una política estática.

La política simétrica utiliza información para clasificar y estimar los nodos como: emisores, receptores y OK. El conocimiento concerniente al estado de los nodos es mantenido por una estructura de datos, comprendida por una lista de emisores, una lista de receptores y una lista OK. La lista es mantenida utilizando un esquema eficiente para manipular acciones tal como mover un nodo de una lista a otra, obtener la lista que le corresponde a un nodo e imponer una sobrecarga constante con respecto al número de nodos en el sistema.

Política de Transferencia. La política de transferencia es una política umbral (threshold), donde las decisiones son basadas sobre la longitud de cola del CPU. La política de transferencia es disparada cuando una nueva tarea se origina o cuando una nueva tarea migra. La política de transferencia hace uso de dos umbrales para clasificar los nodos: un umbral bajo (LT) y un umbral alto (UT). Un nodo se dice que es un emisor si la longitud de la cola > UT, un receptor si la longitud de la cola < LT y un OK

si LT <= la longitud de cola de los nodos <= UT.

Política de Localización. La política de localización tiene los siguientes componentes.

Una Estructura de Datos. La información mantenida en el nodo “i”, consiste de tres listas ordenadas. La primera lista Slisti, contiene los identificadores (id´s) de los nodos que se han identificado como potenciales emisores de tareas. La segunda lista Rlisti contiene los id´s de los nodos que se han identificado como potenciales receptores de tareas. Finalmente la lista Oklisti contiene los id´s de los nodos que no son ni emisores, ni receptores.

Inicialmente se asume que todos los nodos están ociosos y por lo tanto son potenciales receptores. Para reflexionar esta suposición, la lista de cada nodo “i” en un sistema que contiene “n” nodos, son inicilizadas como sigue: Oklisti = null, Slisti = null y Rlisti = i +1, i+2, ...,n, 1,...,i - 1.

Un componente emisor. Cuando la política de transferencia en el nodo “i” determina que el nodo “i” es un emisor, la política de localización en el nodo “i” busca un receptor como sigue:

1. Checa el nodo en la cabeza de Rlisti, se decidió “j”, para determinar si “j” es un receptor.
2. Si “j” responde que es un receptor, retorna sus id’s a la política de transferencia y termina. Si “j” no es un receptor, remueve este de Rlisti y lo agrega al frente de la lista apropiada (Slisti, si “j” es un emisor, Oklisti si “j” es un OK).
3. Para de estar buscando y retorna una falla a la política de transferencia si: a) el nodo observado tiene un ProbeLimiti (un parámetro del algoritmo) sin éxito, b) Rlisti esta vacío, c) “i” no es mayor que un emisor, debido a que se esta complementando la tarea en “i”. En otro caso ir al paso 1.

El nodo observado “j” (probed).

1. Dado que el nodo “i” es observado como un emisor, se debe remover “i” de la lista y se agrega a la cabeza de Slistj. (note que si “i” ya estuvo en Slistj, esta acción simplemente cambia la posición en la lista).

2. Enviar un mensaje de respuesta a “i”, indicando si “j” es un receptor, un emisor o un OK. Si j es un receptor notifica a la política de transferencia que una tarea ha sido agregada (la política de transferencia debe de tomar esto en cuenta, para determinar si “j” continua siendo un receptor).

Un componente receptor. Para obtener un emisor, en un nodo receptor “i”, la política de localización realiza lo siguiente:

1. Observa el nodo seleccionado decimos “j”, para determinar si “j” es un emisor. Los nodos para observar son escogidos de la siguiente forma: a) del frente de Slisti (la información más actualizada es la utilizada primero), b) si Slisti es vacío, entonces en la cola de Oklisti (la información más actualizada es la que se utiliza primero, se espera que el nodo ha cambiado de OK a emisor) c) si Oklisti también esta vacío, de la cola de Rlisti (nuevamente, la información mas actualizada es la que se usa primero).
2. Cuando el nodo “j” responde, remueve “j” de la lista y la inserta en la lista apropiada (Slisti si “j” es un emisor, Oklisti si “j” es un OK y Rlisti si “j” es un receptor). Si “j” es un emisor detenerse. La sobrecarga requerida para esta lista de operaciones es pequeña e independiente del número de nodos en el sistema.
3. Detener la búsqueda y retornar una falla en la política de transferencia si: a) “i” ha alcanzado un ProbeLimit sin éxito, b) todos los nodos que estuvieron en Oklisti y Slisti han sido probados, c) “i” no es un receptor debido al arribo de las tareas. En otro caso ir al paso 1.

El nodo observado “j” (probed)

Si “j” no es un emisor, retorna un mensaje informando a “i” del estado de “j” (receptor u OK). Dado que los “i´s” observados han identificado a “i” como un receptor, remover “i” de cualquier lista e insertarla a la cabeza de Rlistj. Si “j” es emisor, comenzar transfiriendo una tarea a “i”, e informar a “i” que el estado “j” estará después de la transferencia.

Política de Selección. El componente emisor considera solamente tareas nuevas para ser transferidas. El componente receptor, puede transferir tareas nuevas sin estado de ejecución (no-preemptives) así

como tareas con estado de ejecución (preemtives).

Política de Información. La política de información es sobre demanda (demand-driven), debido a que la actividad de estimación de carga comienza cuando el nodo llega a ser un emisor o un receptor.

Discusión. En cargas altas, la probabilidad de que un nodo este descargado es casi imposible resultando una estimación de carga no exitosa para un componente emisor. En los sistemas de baja carga, la estimación de carga receptora generalmente falla, estas fallas no atenúan el desempeño por necesitar capacidades de procesamiento extra, ya que esta disponible en sistemas de baja carga. Es decir utilizar la compartición de carga emisora en sistemas de baja carga, compartición de carga receptora en cargas altas y la compartición de carga iniciada simétricamente, para cargas moderadas.

3.10 Conclusiones.

Habiendo considerado la taxonomía, requerimientos y componentes de los algoritmos de balanceo de carga en el capitulo 2. En este capitulo ilustramos algunos algoritmos que se han estudiado en la literatura, y que son una muestra representativa de la clasificación jerárquica analizada en el capitulo 2, los algoritmos que no pueden ser clasificados jerárquicamente son clasificados en forma plana (flat), dentro de esta clasificación tenemos las estrategias emisoras o receptoras. Este tipo de estrategias pueden usarse para determinar que computadora puede iniciar el balanceo de carga. En las estrategias emisoras, la actividad de distribución de carga es iniciada por un nodo sobrecargado (emisor) que intenta enviar una tarea a un nodo descargado (receptor). Se discutieron tres algoritmos emisores simples pero aún efectivos, estudiados por Eager, Lazowska y Zohorjan [Eager 86]. Los algoritmos son el Aleatorio, Cola Más Corta y Umbral. En los algoritmos receptores, la actividad de distribución de carga es iniciada de un nodo descargado (receptor) que intenta obtener una tarea de un nodo sobrecargado (emisor) [Eager 86]. En los algoritmos iniciados simétricamente, ambos emisor y receptor buscan un emisor o un receptor para transmitir una tarea, este algoritmo tiene las ventajas del emisor y del receptor. Cuando la carga es baja el emisor es más exitoso para obtener un nodo descargado. Cuando la carga del sistema es alta, el receptor es más exitoso para obtener un nodo sobrecargado. Sin embargo estos algoritmos no son inmunes a las desventajas que ofrecen los algoritmos emisores y receptores. Un algoritmo iniciado simétricamente puede ser construido

utilizando las políticas de localización y de transferencia de los dos algoritmos (emisor y receptor). Otro algoritmo simétrico es el algoritmo llamado Above-Average [Singhal 94], que es el que se describió en el capitulo. Este capitulo concluye ilustrando el trabajo de Shivaratri y Krueger [Krueger 94] el cual proponen dos políticas de planeación adaptativas usadas en un algoritmo simétricamente estable.

Capítulo 4. Estrategias y Arquitecturas para Balancear Carga en CORBA.

Las estrategias en CORBA para diseñar un servicio de balanceo de carga pueden ser: *Por-sesión, Por-requerimiento y Sobre-demanda.* Las políticas para implementar dicho servicio pueden ser clasificadas dentro de las siguientes dos categorías: *Adaptativas y No-adaptativas.* Este capítulo tiene como principal objetivo ilustrar cada una de las *Arquitecturas de Balanceo de Carga* que podemos diseñar en CORBA combinado las tres estrategias anteriores en conjunción con las dos políticas, tal como lo menciona Ossama y Douglas C. Schmidt en [Ossama 01a, 01b, 01c, 03a y 03b]. El capítulo concluye mencionando los elementos estructurales que debe tener todo sistema de balanceo de carga diseñado en CORBA con el propósito de tener una mejor idea de cómo la implementación de balanceadores de carga debería de ser vista para operar [IONA 02].

4.1 Introducción.

Los mecanismos de balanceo de carga distribuyen equitativamente la carga de trabajo de los clientes entre un conjunto de servidores para mejorar el tiempo de respuesta global del sistema. Los mecanismos de balanceo de carga generalmente se encuentran en uno de los niveles siguientes [Ossama 01a].

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

Balanceo de Carga Basado en Red. Los Servidores de Nombres de Dominio (DNS) y los ruteadores IP que sirven a una gran cantidad de máquinas host proveen este tipo de balanceo de carga. Por ejemplo cuando un cliente resuelve un hostname, el DNS puede asignar una dirección IP a cada requerimiento basado en las condiciones de carga actual, el cliente entonces contacta el servidor designado, sin que el cliente se entere, un servidor diferente puede ser designado en la próxima resolución de nombres. El balanceo de carga en ésta capa es algo limitado, ya que no toman en cuenta el contenido de los requerimientos del cliente por encontrarse a nivel de red (ver Figura 4.1), el balanceo de carga a nivel de red tiene la desventaja de que las decisiones de balanceo de carga son

basadas sobre el destino de los requerimientos más que el contenido de los requerimientos.

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

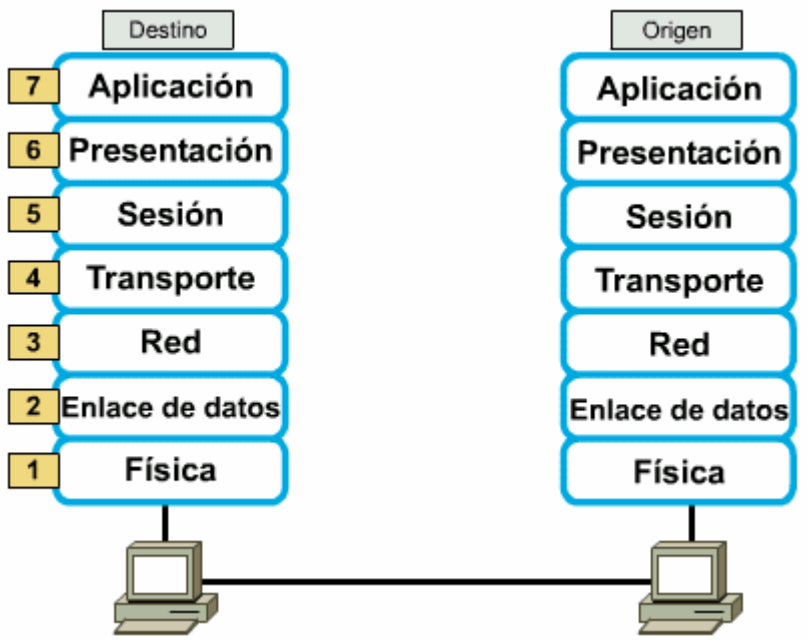


Figura 4.1 El Modelo OSI.

Balanceo de Carga Basado en Middleware. Las interacciones globales son acopladas por las arquitecturas llamadas Middleware. La arquitectura más común de Middleware para aplicaciones orientadas a objetos distribuidos es Common Object Request Broker Architecture (CORBA) [OMG 01]. 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.

4.2 Conceptos Claves en un Servicio de Balanceo de Carga Basado en CORBA.

Los conceptos claves de un servicio de balanceo de carga basado en CORBA son mostrados en la Figura 4.2 y son descritos como sigue [Ossama 01a, Balasubramanian 04]:

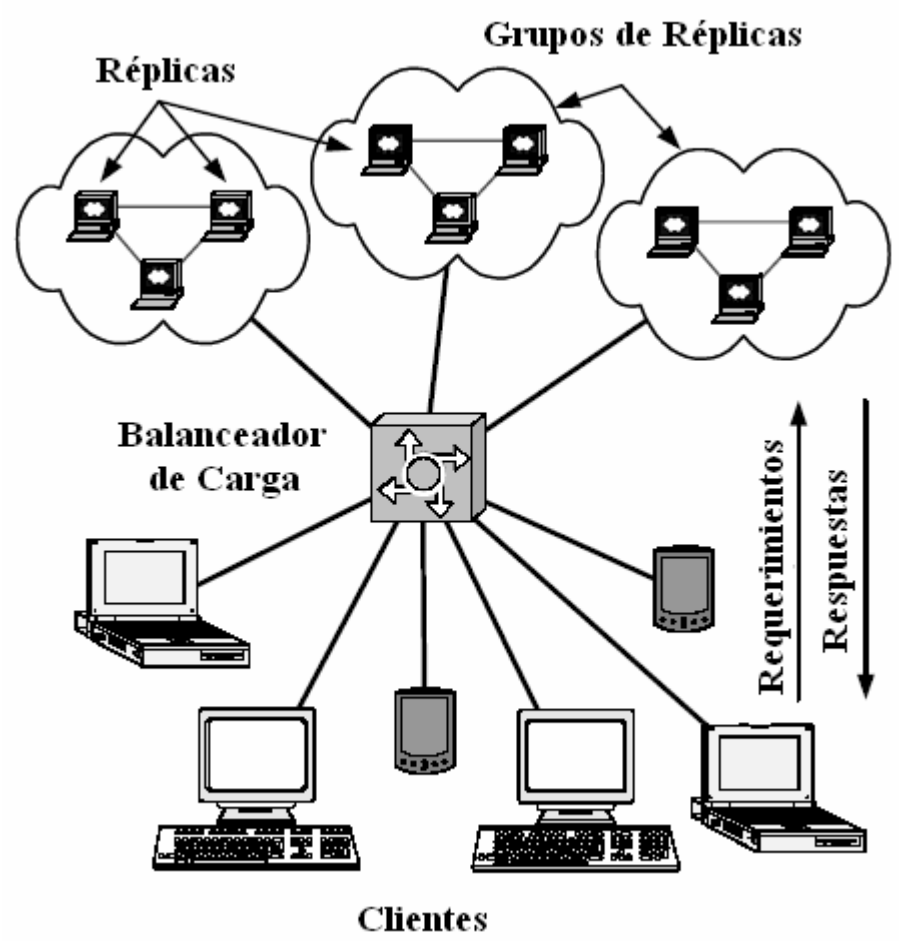


Figura 4.2 Conceptos Claves en un Servicio de Balanceo de Carga Middleware [IONA 02].

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. Esta 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 se define como el periodo de tiempo que un cliente invoca operaciones remotas (requerimientos) para acceder servicios proporcionados por objetos

en un servidor particular (ver requrimientos y respuestas en Figura 4.2).

4.2.1 Políticas de Balanceo de Carga en CORBA.

Cuando se diseña un servicio de balanceo de carga en CORBA es importante seleccionar una estrategia adecuada que decida que réplica procesará el requerimiento que llega. En general, las políticas de balanceo de carga en CORBA pueden ser clasificadas dentro de las categorías siguientes [Ossama 01a]:

- *No adaptativas*: Un balanceador de carga puede usar políticas no adaptativas para el atado de requerimientos y esa política es aplicada por toda la vida del cliente. El algoritmo seleccionado puede ser tan simple como el Round-Robin o el Random para seleccionar una réplica en donde se procesará el requerimiento.

- *Adaptativas*: Un balanceador de carga puede usar políticas adaptativas que utilicen información en tiempo de ejecución (réplicas disponibles, carga de trabajo en las réplicas, requerimientos en espera, etc.) para seleccionar la mejor réplica, que procese sus requerimientos, o bien, cambiar a otra réplica cuando ésta no proporcione el resultado esperado por el sistema.

4.2.2 Modelo del Sistema Distribuido.

La Figura 4.3 muestra el modelo del sistema distribuido usado como base para los experimentos de balanceo de carga usados en esta tesis [Balasubramanian 04].

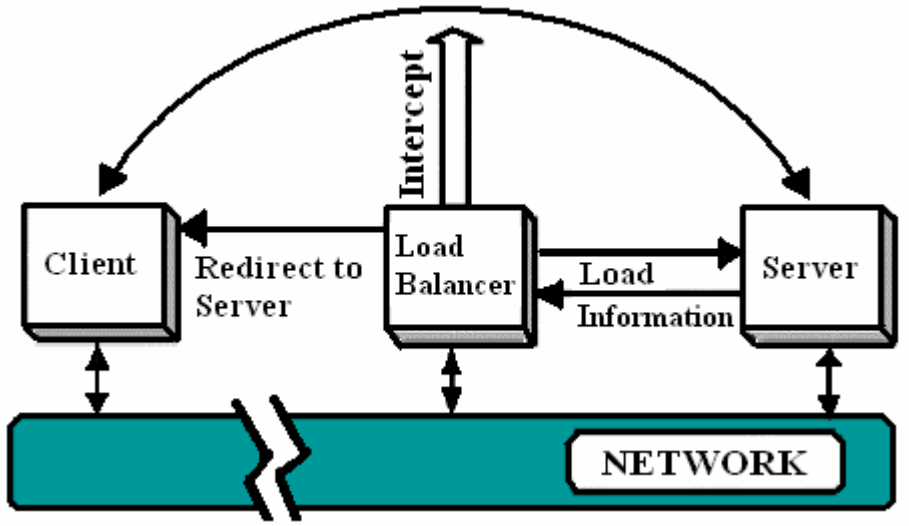


Figura 4.3 Modelo de un Sistema Distribuido [Balasubramanian 04].

En este modelo, los sistemas distribuidos son construidos interconectando clientes y servidores. Cada

servidor es un miembro del grupo de objetos. Todos los servidores se registran ellos mismo con el balanceador de carga, que consisten de uno o más servidores que median entre los diferentes elementos del servicio de balanceo de carga. Cuando un cliente invoca una operación, estos requerimientos son inicialmente interceptados por el balanceador de carga, el balanceador de carga obtiene una replica del grupo de objetos (por ejemplo un servidor) situable para manejar los requerimientos y entonces redirecciona los requerimientos a la replica designada usando el soporte de re-direccionamiento de requerimientos de middleware. Las aplicaciones del cliente son de esta manera asignadas a una réplica del grupo para manejar sus requerimientos.

4.3 Estrategias para Balancear Carga en CORBA.

Hay varias estrategias para diseñar en CORBA un servicio de balanceo de carga [Ossama 01a]. 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 el atado del requerimiento a una réplica cada vez que es invocado. La arquitectura se define por el periodo de vida del requerimiento.
- *Sobre-demanda*: Los requerimientos del cliente son enviados en conjuntos y atados a una réplica seleccionada de acuerdo al algoritmo de balanceo de carga que toma como base el estado global del sistema. La arquitectura de balanceo se define por el atado de un conjunto de requerimientos.

4.4 Arquitecturas de Balanceo de Carga en CORBA.

Combinando las estrategias descritas anteriormente de varias maneras y las políticas, es posible crear las siguientes arquitecturas de balanceo de carga [Ossama 01a].

4.4.1 Arquitectura No Adaptativa Por-Sesión.

Una manera de diseñar una Arquitectura de balanceo de carga en CORBA es hacer que el balanceador de carga seleccione la réplica destino donde una sesión *Cliente/Servidor* quedará establecida, es decir, cuando un cliente obtiene una referencia de objeto a un objeto CORBA (nombre de la réplica) y se conecta a ese objeto. Tal como se muestra en la Figura 4.4.

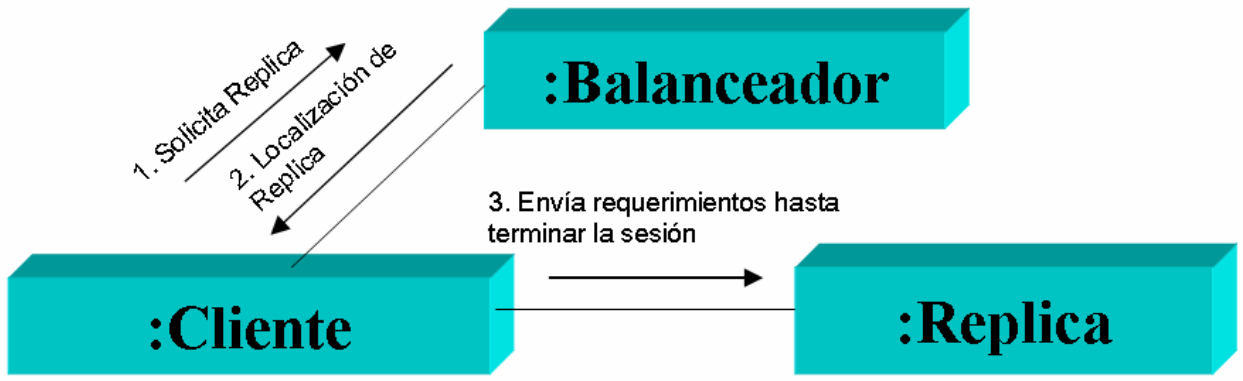


Figura 4.4 Arquitectura de Balanceo de Carga No Adaptativa Por-Sesión.

Note que la política de balanceo en esta arquitectura es *No adaptativa* ya que el cliente interactúa con el mismo servidor al cual fue atado originalmente. Esta arquitectura es adecuada para la política de balanceo de carga que implementa algoritmos Round-Robin o Random.

4.4.2 Arquitectura No Adaptativa Por-Requerimiento.

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

4.4.3 Arquitectura No Adaptativa Sobre-Demanda.

Tiene las mismas características Por-sesión. Sin embargo, la arquitectura *No Adaptativa* Sobre-demanda permite reorganizar los requerimientos del cliente de una manera arbitraria en el tiempo. La

información en tiempo de ejecución, como: réplicas disponibles, carga de trabajo en las réplicas, requerimientos en espera, etc., no es usada para decidir cuándo se reorganizan los requerimientos de los clientes. En lugar de eso, los requerimientos de los clientes se pueden reorganizar en intervalos de tiempo.

4.4.4 Arquitectura Adaptativa Por-Sesión.

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

4.4.5 Arquitectura Adaptativa Por-Requerimiento.

La Figura 4.5 muestra una arquitectura de balanceo de carga *Adaptativa* Por-requerimiento, este diseño introduce un servidor “front-end”, que es un proxy [Ossama 01a], que recibe todos los requerimientos de los clientes. En este caso, el servidor “front-end” es el balanceador de carga. El balanceador de carga selecciona una réplica servidor apropiada “back-end” acorde con las políticas de balanceo de carga y envía los requerimientos a la réplica. El servidor proxy “front-end” espera la respuesta de las réplica y la envía de regreso a los clientes.

El principal beneficio de una arquitectura Adaptativa Por-requerimiento es el potencial para la gran escalabilidad y transparencia. Por ejemplo, el servidor proxy puede examinar la carga actual en cada réplica y después seleccionar el destino de cada requerimiento permitiendo distribuir la carga más equitativamente. Esta arquitectura es adecuada para usar políticas adaptables de balanceo de carga. Desafortunadamente, esta arquitectura también puede introducir excesivos retrasos y sobrecargas en la red, debido a que en esta arquitectura se usarían cuatro mensajes en la red:

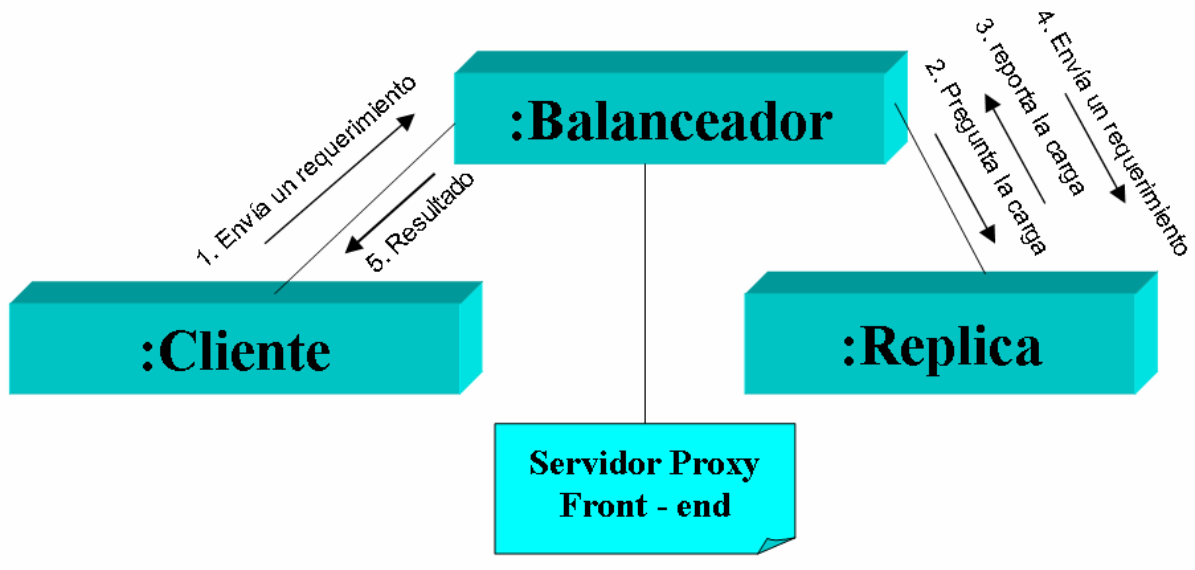


Figura 4.5 Arquitectura de Balanceo de Carga Adaptativa Por-requerimiento.

- El envío del requerimiento al servidor “fron-end”,
- El envío del requerimiento del “front-end” a la réplica (“back-end”),
- La respuesta del “back-end” al “front-end”, y
- La respuesta correspondiente del “front-end” al cliente.

4.4.6 Arquitectura Adaptativa Sobre-Demanda.

Esta arquitectura se muestra en la Figura 4.6 el cliente recibe una referencia de objeto del balanceador de carga. Usando un mecanismo estándar de CORBA, el balanceador de carga puede redirigir el requerimiento del cliente a la réplica apropiada. El cliente continuará usando el nuevo objeto de referencia y se comunicará con la réplica directamente hasta que sea redireccionado otra vez o terminen su sesión. Usando la información de carga actual y la política especificada (*adaptativa o no adaptativa*), el balanceador de carga puede determinar distribuir la carga de manera equilibrada.

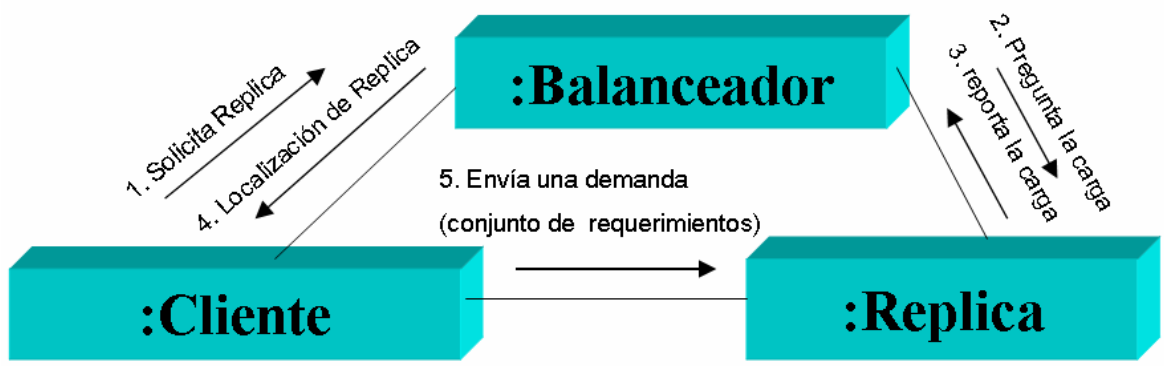


Figura 4.6 Arquitectura de Balanceo de Carga Adaptativa Sobre-Demanda..

4.5 Elementos Estructurales de un Sistema de Balanceo de Carga.

Los componentes de los balanceadores de carga funcionales son descritos a continuación para proveer una mejor idea de cómo la implementación de balanceadores de carga bajo CORBA debería ser vista para operar [IONA 02].

1. Localizador de Réplicas.
2. Monitor de Carga.
3. Analizador de Carga.
4. Balanceador de Carga.

4.5.1 Localizador de Réplicas.

Un sistema de balanceo de carga debe tener un mecanismo para redirigir los requerimientos del cliente a una réplica apropiada. Idealmente el código de la aplicación del cliente pudiera no ser modificado para soportar tal funcionalidad. Aplicaciones basadas en CORBA contienen las construcciones necesarias para soportar transparentemente el envío de requerimientos del cliente. Una manera de hacer transparente el envío de los requerimientos del cliente es utilizando LOCATION_FORWARD implementado en CORBA (ver Figura 4.7) [IONA 02].

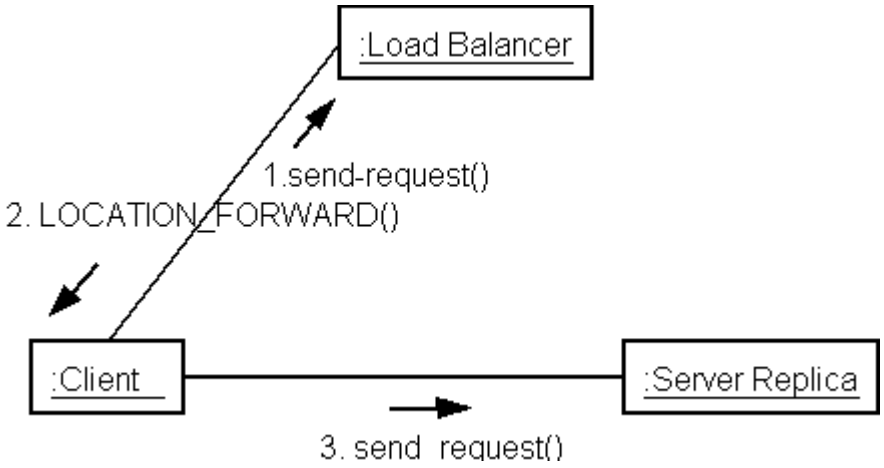


Figura 4.7 Direccionamiento Transparente en el Envío de Requerimientos del Cliente [IONA 02].

4.5.2 Monitor de Carga.

Este elemento estructural provee la habilidad de monitorear carga sobre una réplica dada, además de responder a los requerimientos de control de balanceo de carga hechos por el balanceador de carga.

La función principal de este componente es la de:

1) Monitorear la carga sobre una réplica dada.
2) Reportar la carga de las réplicas al balanceador de carga.

Como describimos en la Figura 4.8, un monitor de carga debe ser configurado con una de las dos políticas siguientes:

- **Pull policy.** En este modo, un balanceador de carga puede preguntar al monitor de carga la carga de una réplica dada.

- **Push policy**. En este modo, el monitor de carga puede enviar un reporte de carga al balanceador de carga (Push).

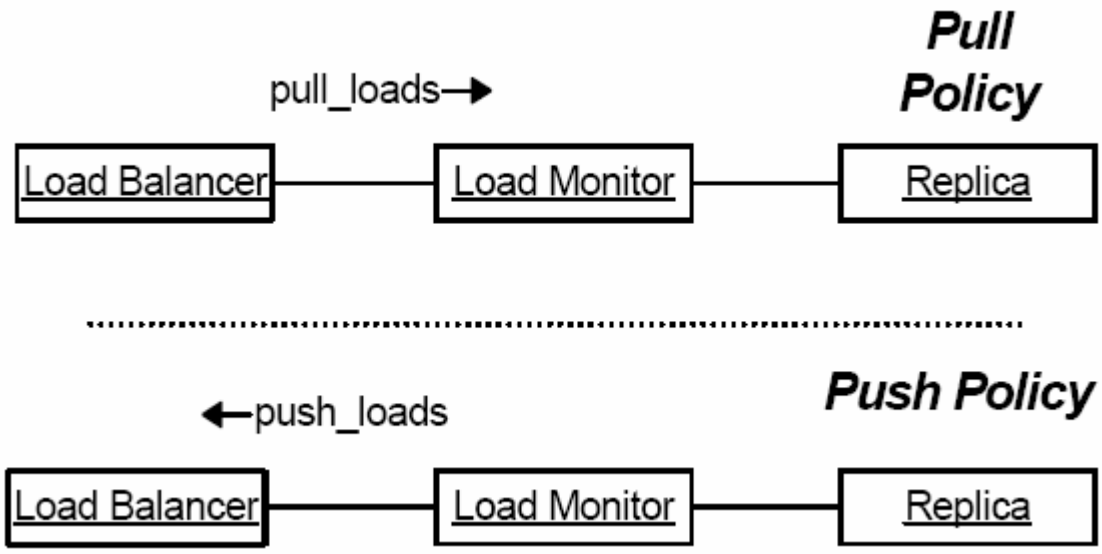


Figura 4.7 Políticas del Monitor de Carga [IONA 02].

4.5.3 Analizador de Carga.

En general, no es común que todas las aplicaciones distribuidas exhiban las mismas condiciones de carga, significa que algunas estrategias de balanceo de carga 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 de carga. Esas estrategias son encapsuladas en el componente analizador de carga [Ossama 01a,b], [Balasubramanian 04].

La propuesta primaria del analizador de carga, es tan evidente como su nombre, éste determina la estrategia de balanceo de carga que se va a implementar dependiendo de la aplicación (ver Figura 4.8). Un soporte de múltiples estrategias de balanceo de carga deben de estar disponibles en el analizador de carga. Por ejemplo, una estrategia de balanceo de carga diferente debe ser empleada por cada grupo de réplicas registrado. Dado que el conocimiento a priori de los requerimientos de una aplicación distribuida no siempre está disponible, es importante tener la habilidad para dinámicamente agregar o remplazar estrategias de balanceo de carga soportadas por algún analizador de carga.

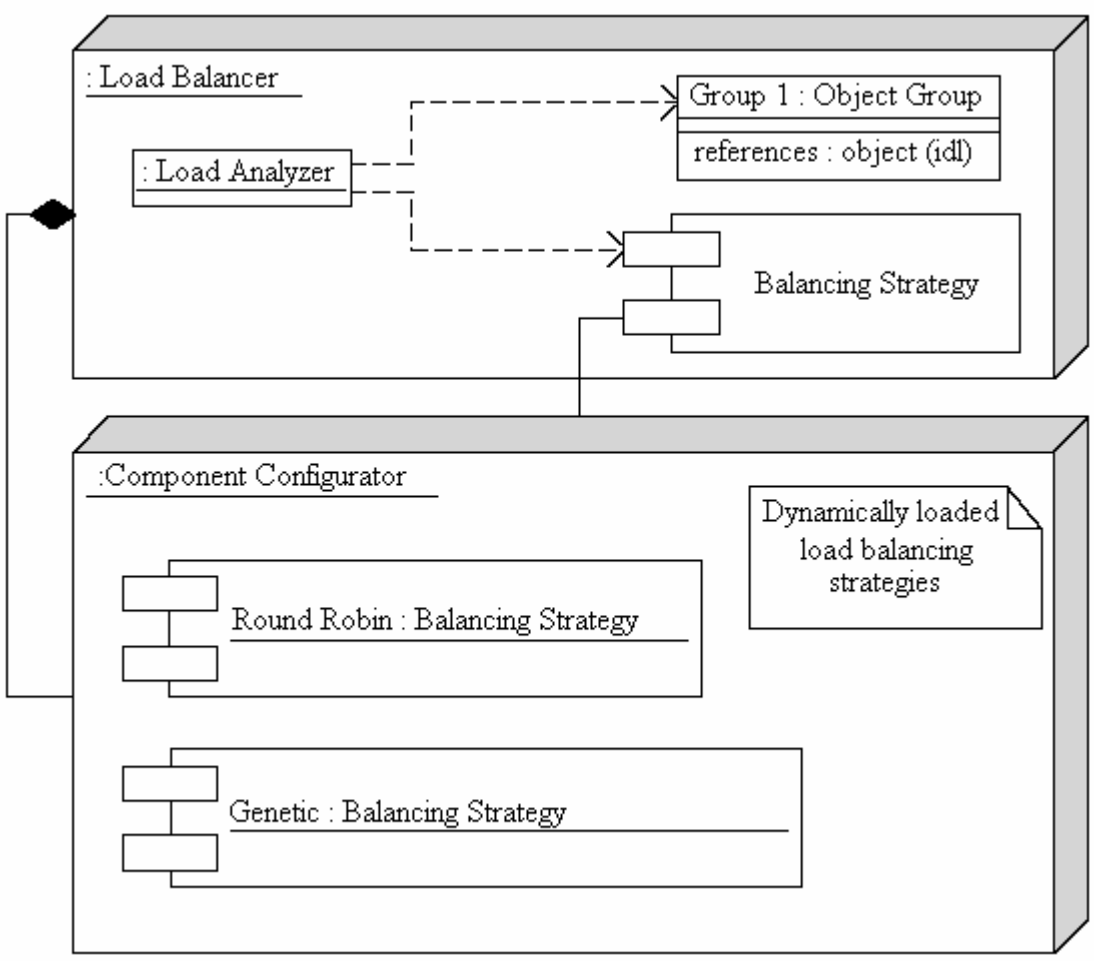


Figura 4.8 Diferentes Estrategias para Balanceo de Carga [IONA 02]

4.5.4 Balanceador de Carga.

Dados todos los elementos estructurales descritos anteriormente, es necesario coordinar y mediar sus tareas individuales para proveer la funcionalidad de un balanceador de carga global (ver Figura 4.2). Aquí, el sistema de balanceo de carga, como un todo, básicamente implementa un patrón de diseño mediador. Tal diseño asegura que las tareas concretas sean encapsuladas con componentes especializados, que proveen los siguientes beneficios:

- Simplificación del diseño global.
- Maximizar la transparencia dado que las entidades fuera del balanceo de carga pueden permanecer olvidadas al comportamiento interno del balanceador, además permite a los

componentes internos olvidarse de otros componentes en el sistema de balanceo de carga.

4.6 Conclusiones.

La carga puede ser balanceada en varios niveles: a nivel de Red, a nivel Sistema Operativo y Middleware. Las arquitecturas de balanceo de carga basadas en la Red y Sistemas Operativos sufren de varias limitaciones tales como: la carencia de flexibilidad que surge de la inhabilidad de soportar métricas a nivel aplicación cuando hacemos decisiones de balanceo de carga, la carencia de adaptabilidad que ocurre debido a la ausencia de retroalimentación relaciona con la carga de un conjunto dado de objetos. Las arquitecturas de balanceo de carga basadas en middleware, específicamente las que se basan sobre el estándar de CORBA, han sido ideadas para sobreponerse a esas limitaciones. Este capitulo describo las arquitecturas que debemos tomar en cuenta cuando diseñamos un servicio de balanceo de carga bajo CORBA. Primero ilustramos los conceptos claves, políticas, estrategias y modelo distribuido de un servicio de balanceo de carga middleware bajo CORBA siguiendo a Balasubramanian J. en [Balasubramanian 04] y posteriormente ilustro las arquitecturas de balanceo de carga que podemos diseñar en CORBA combinando las tres estrategias anteriormente mencionas en conjunción con las políticas acorde con Ossama O. en [Ossama 01a, b]. El capítulo concluyo mencionando los elementos estructurales que debe tener todo sistema de balanceo de carga diseñado en CORBA según IONA Technologies en [IONA 02].

Capítulo 5. Diseño de la Arquitectura para el Balanceo Dinámico de Carga Bajo CORBA.

Este capítulo tiene como objetivo mostrar el análisis, diseño y construcción de la arquitectura que permite el balanceo dinámico de carga por demanda adaptativa en ambientes distribuidos orientados a objetos. La arquitectura se apoya en el Lenguaje de Modelado Unificado (UML) [Booch 94, Booch 99, Fowler 97], en el estándar de objetos distribuidos CORBA [OMG 99, 01, 02] y en el lenguaje de definición de interfaces JAVA-IDL, que permite la integración del lenguaje de programación JAVA en aplicaciones distribuidas CORBA [OMG 99, Vogel 98].

5.1 Análisis de la Arquitectura.

Según Mellor el propósito del análisis es proporcionar una descripción del problema [Booch 94]. La descripción debe de ser concreta, consistente, legible y revisable, por las diversas partes interesadas y ser comparada con la realidad.

Para efectuar el análisis, Booch [Booch 94], establece que el análisis de requerimientos y el análisis del dominio del sistema, son dos pasos que deben ser liberados durante esta etapa. Por una parte el análisis de los requerimientos del sistema debe proporcionar la escritura básica de las funciones que el sistema deberá realizar y por su parte el análisis del dominio del sistema proporciona las estructuras lógicas que son claves para el sistema.

5.1.1 Análisis de Requerimientos para Implementar la Arquitectura de Balanceo de Carga.

5.1.1.1 Declaración del Problema.

La arquitectura consiste de “N” computadoras heterogéneas que funcionan como réplicas, una por cada computadora, una computadora analizador de carga que mantiene comunicación directa con la base de datos, otra computadora que funciona como monitor de carga y “N” computadoras más que funcionan como clientes. Todas las computadoras están conectados bajo el estándar de CORBA que implementa SUN Microsystems en su producto JAVA-IDL versión 1.3 y posteriores. Los clientes y el monitor se comunican con el analizador de carga y las réplicas utilizando objetos distribuidos CORBA

a través del canal de comunicación ORB (Object Request Broker) como se puede observar en el diagrama de la Figura 5.1.

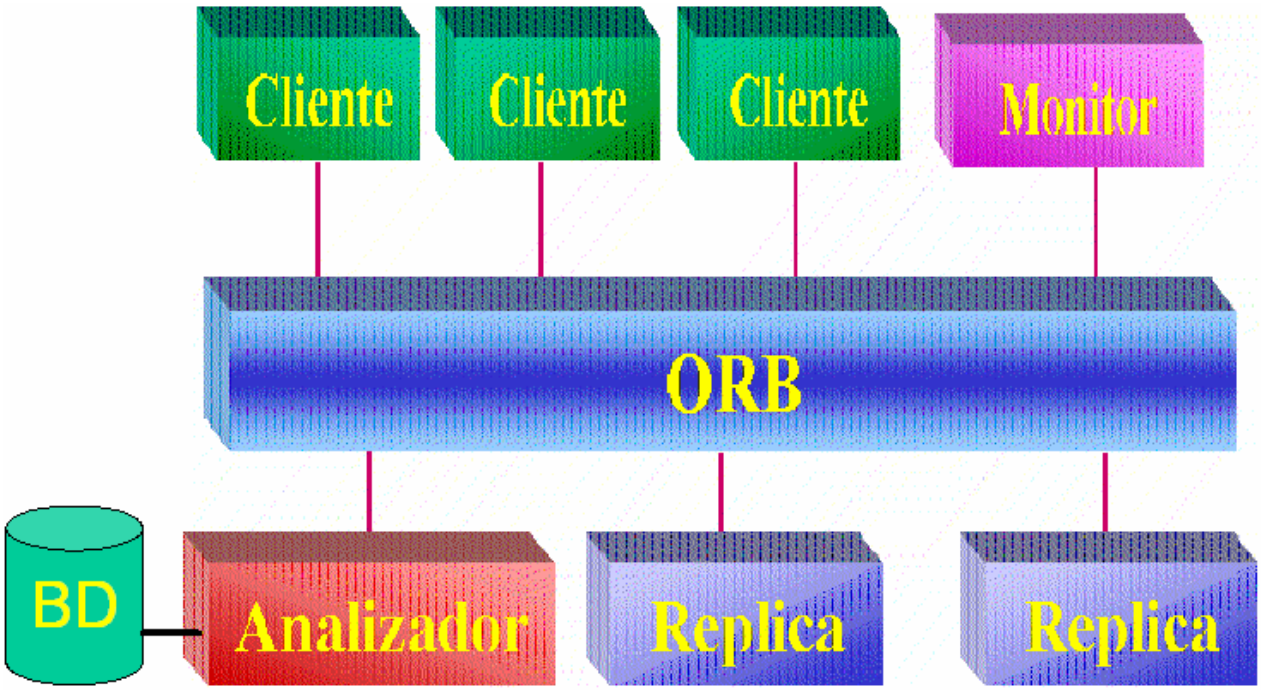


Figura 5.1 Componentes de la Arquitectura.

La descripción rápida del contenido de la arquitectura es la siguiente.

1. **LBA_Analizador.** Este componente tiene la responsabilidad de recibir, procesar y controlar los requerimientos de los clientes. Además debe seleccionar la réplica donde se procesarán los requerimientos, tomando como base la estrategia de balanceo de carga utilizada por la arquitectura. Este componente es el único que tiene contacto directo con la base de datos.
2. **LBA_Réplica.** Componente encargado de procesar los requerimientos de los clientes. Son los servidores que proporcionan respuesta a los requerimientos de los clientes.
3. **LBA_Cliente.** Componente que solicita los requerimientos en la arquitectura y pide que se le dé servicio.
4. **LBA_bd.** Es la base de datos encargada de almacenar la información necesaria para la toma de decisiones y formar estadísticas del proceso. Además, contiene la información de estado, las características y tipos de réplicas.

5. **LBA_Monitor.** Componente encargado de estimar la carga de las réplicas, este componente utiliza una política de monitoreo periódica (política poll) en estrategias que checan el estado en tiempo de ejecución.

Además el componente *LBA_Analizador* contendrá diferentes estrategias de balanceo de carga. Este es el componente más importante en nuestra arquitectura, ya que puede utilizar diversas políticas por demanda *Adaptativas y No-Adaptativas.*

- *No Adaptativas*: La arquitectura puede usar una política *No Adaptativa* para el atado de requerimientos y esa política es aplicada por toda la vida del cliente. La estrategia seleccionada puede ser tan simple como la estrategia Round-Robin o la Aleatoria (Random) para seleccionar una réplica en donde se procesará el requerimiento.

- *Adaptativas*: Un balanceador de carga puede usar políticas *Adaptativas* que utilicen información en tiempo de ejecución (réplicas disponibles, carga de trabajo en las réplicas, requerimientos en espera, etc.) para seleccionar la mejor réplica, que procese sus requerimientos, o bien, cambiar a otra réplica cuando ésta no proporcione el resultado esperado por el sistema. En este tipo de política utilizamos las estrategias Umbral, Cola Más Corta (C.M.C.) y Genética.

5.1.2 Análisis del Dominio para Implementar la Arquitectura de Balanceo de Carga.

En esta sección se muestran los diagramas de clases, secuencia, cooperativo y despliegue correspondientes a la arquitectura de balanceo de carga.

5.1.2.1 Diagrama de Clases.

Un diagrama de estructura estática (diagrama de clases) muestra el conjunto de clases y objetos que son parte de un sistema, junto con sus relaciones existentes entre estas clases y objetos [Booch 99].

La Figura 5.2 muestra el diagrama de clases de la arquitectura de balanceo de carga, en él observamos claramente que la clase *Balanceador*, está compuesta de las clases *LBA_Réplica*, *LBA_Monitor*, *LBA_Cliente y LBA_Analizador*. Estas son las clases principales que conforman la arquitectura de balanceo de carga. Por otra parte la clase *LBA_Analizador* utiliza la clase *Estrategia* para establecer la estrategia de carga que utilizará la arquitectura, la estrategia que se establece dependerá de la política seleccionada. Si se selecciona una política No-Adaptativa se podrán utilizar las estrategias: Round-Robin y Aleatoria. En caso contrario, si se elige una política Adaptativa las estrategias utilizadas serán: Umbral, CMC o Genética.

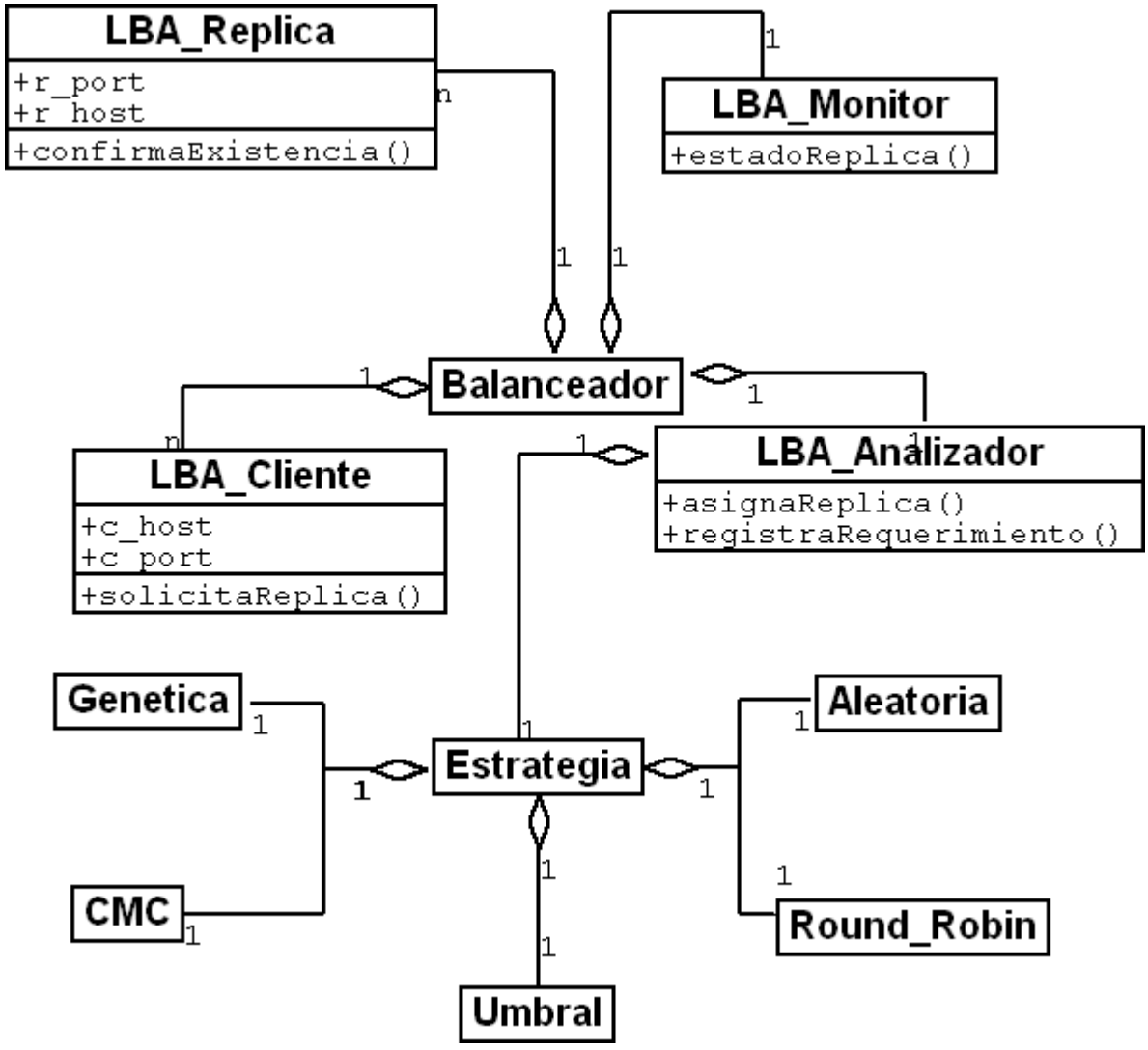


Figura 5.2 Estructura Estática del Servicio de Balanceo de Carga.

5.1.2.2 Diagrama de Secuencia.

Los diagramas de interacción son modelos que describen la manera en que colaboran grupos de objetos para cierto comportamiento [Fowler 97]. La arquitectura de balanceo de carga fue desarrollada basada en este tipo de diagramas. El diagrama de la Figura 5.3 muestra la secuencia de operaciones que ocurre cuando el balanceador de carga obliga a una réplica sobrecargada a rechazar la demanda de requerimientos de los clientes y a su vez elige una réplica que esté en posibilidades de aceptar esta demanda.

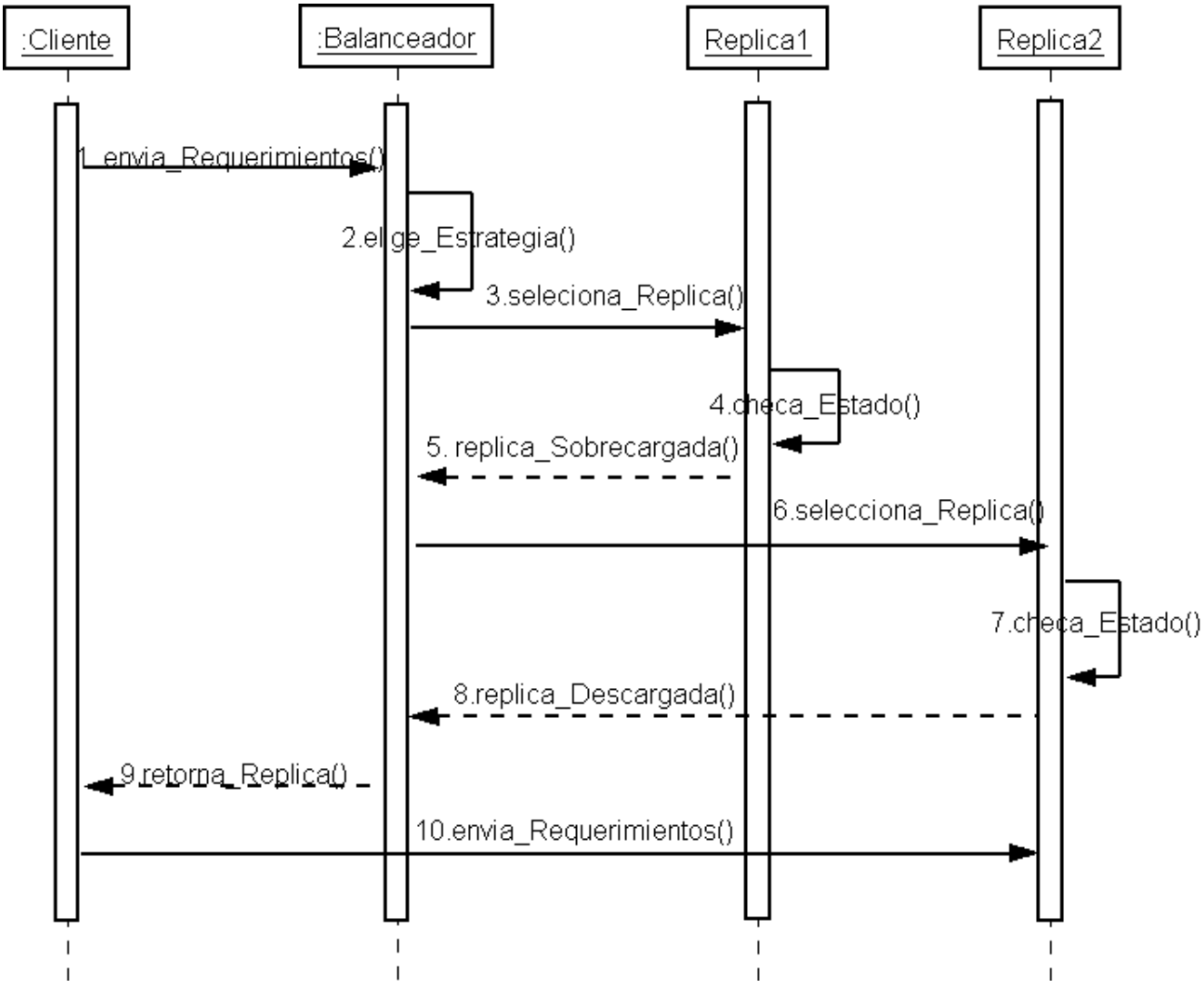


Figura 5.3 La Arquitectura de Balanceo de Carga Elige una Réplica Donde Procesar Requerimientos.

1. El cliente solicita un requerimiento. El requerimiento es interceptado transparentemente por el balanceador de carga.
2. El balanceador de carga elige la estrategia de balanceo de carga, que se utiliza para seleccionar la réplica.
3. Una vez elegida la estrategia el balanceador de carga selecciona una réplica para transferir carga.
4. Se checa el estado actual de la réplica seleccionada.
5. Si la réplica seleccionada no está en posibilidades de almacenar los requerimientos, se retorna un mensaje en donde se especifica que la réplica se encuentra sobrecargada.
6. El balanceador de carga vuelve a elegir una nueva réplica, aplicando nuevamente la estrategia de balanceo de carga.
7. Se vuelve a checar el estado actual de la nueva réplica seleccionada.
8. Si la réplica se encuentra en posibilidades de procesar requerimientos de los clientes, se retorna un mensaje indicando esta situación.
9. Una vez que el balanceador de carga encuentra la réplica, se le indica al cliente quien deberá atender sus peticiones de requerimientos.
10. El cliente solicita a la réplica que le atienda sus requerimientos.

5.1.2.3 Diagrama de Colaboración.

Un diagrama de colaboración es un diagrama de iteración que resalta la organización estructural de los objetos que envían o reciben mensajes. Un diagrama de colaboración muestra un conjunto de objetos, enlaces entre esos objetos y mensajes enviados y recibidos por estos objetos [Booch99]. Los diagramas de colaboración se utilizan para describir la vista dinámica del sistema.

En la Figura 5.4 se muestra un diagrama cooperativo de la arquitectura de balanceo de carga, en él se muestra como el objeto cliente le solicita al analizador de carga una réplica para poder procesar sus requerimientos. El monitor por su parte checa la carga del estado actual de todas las réplicas que se encuentra monitoreando y la entrega al analizador de carga, después que el monitor entrega dicha información, el analizador se comunica con la base datos con el propósito de tomar alguna decisión o

mantener actualizada la información. Cuando se asigna una réplica al cliente, este se mantiene en comunicación con la réplica hasta que termina la petición de sus requerimientos o hasta que el analizador de carga designe una nueva réplica.

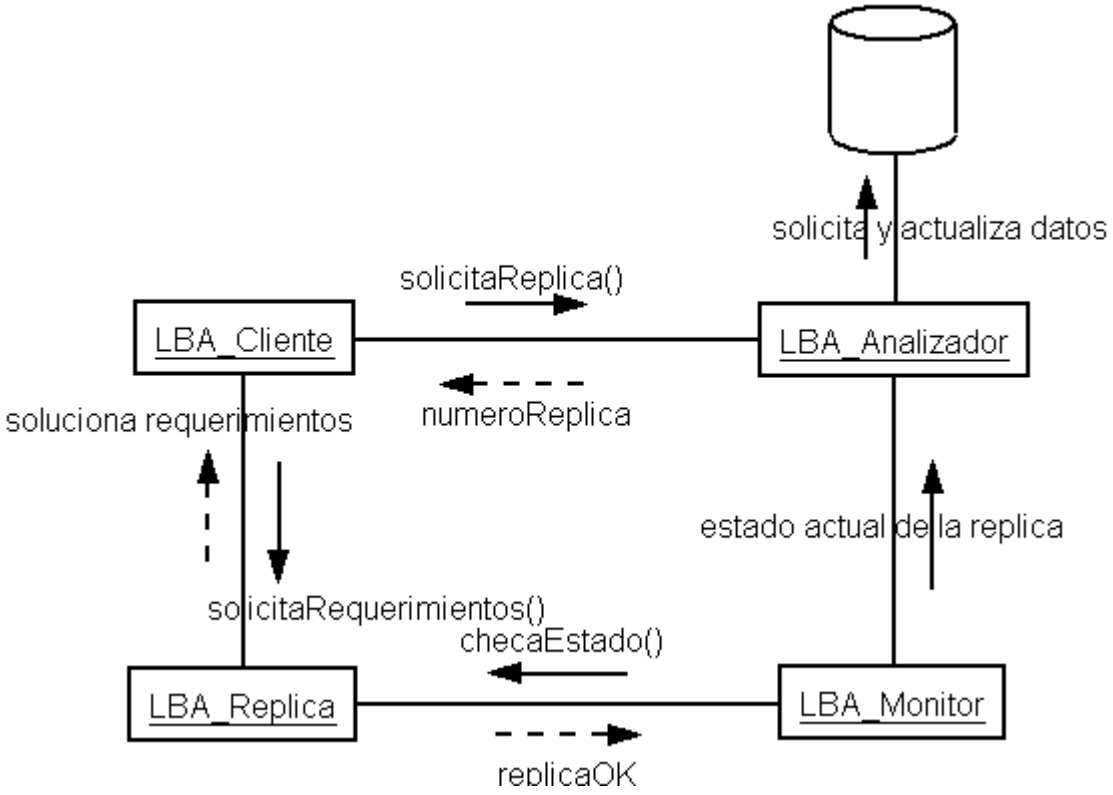


Figura 5.4 El Cliente Solicita una Réplica para Procesar sus Requerimientos

5.2 Diagrama de Despliegue.

El diagrama de despliegue es aquel que muestra las conexiones físicas entre los componentes de software y de hardware en el sistema. Así, el diagrama de despliegue es un buen sitio para mostrar cómo se enrutan y se mueven los componentes y los objetos, dentro de un sistema distribuido. Cada nodo de un diagrama de despliegue representa alguna clase de unidad de cómputo; en la mayoría de los casos se trata de una pieza de hardware. El hardware puede ser un dispositivo o un sensor simple, o puede tratarse de un mainframe. La Figura 5.5 muestra PC´s conectadas por medio de TCP/IP. Las conexiones entre nodos muestran las rutas de comunicación a través de las cuales interactuará el sistema. La figura muestra el corazón de la arquitectura y describe lo siguiente:

1. Lba_Cliente. Solicita al Balanceador de Carga una réplica para procesar sus requerimientos.
2. El Balanceador. Pide al LBA_Analizador que seleccione la réplica apropiada.
3. LBA_Analizador se comunica con LBA_Monitor para estimar la carga de cada una de las réplicas.
4. LBA_Monitor. Checa el estado de cada una de las réplicas y envía el resultado a LBA_Analizador.
5. LBA_Analizador. Elige la réplica adecuada, según la estrategia de planeación que tenga en turno, una vez seleccionada la réplica, envía la ubicación de la réplica a LBA_Cliente.
6. LBA_Cliente se comunica con LBA_Réplica y solicita el servicio a sus requerimientos. El LBA_Cliente se mantiene en comunicación con LBA_Réplica hasta que termine de solicitar requerimientos o hasta que el LBA_Analizador lo reubique nuevamente.

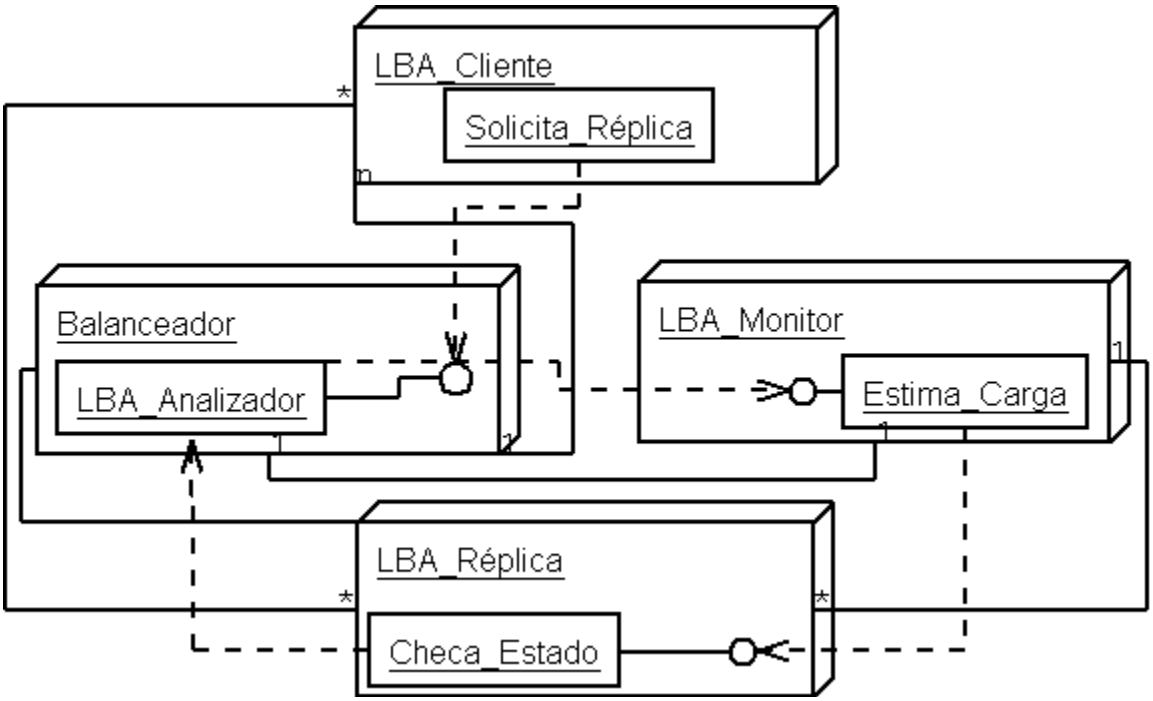


Figura 5.5 Servicio de Balanceo de Carga.

5.3 El Modelo del Sistema.

Actualmente la arquitectura cuenta con 9 PC´s dedicadas a los clientes tienen sistema operativo

Windows XP, 128 Mbytes en RAM y son Pentium IV, 4 computadoras para las réplicas, una réplica mantiene las mismas características que sus clientes, 2 réplicas tienen sistema operativo Linux Mandrake versión 9.0, una tiene 256 Mbytes en RAM con procesador Pentium III en 850Ghz, la otra tiene 128Mbytes en RAM y es Pentium IV en 2.6Ghz. La última réplica es un SUN NETRA X1 con sistema operativo Solaris versión 8.0. 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 1.3 y posteriores (ver Figura 5.6).

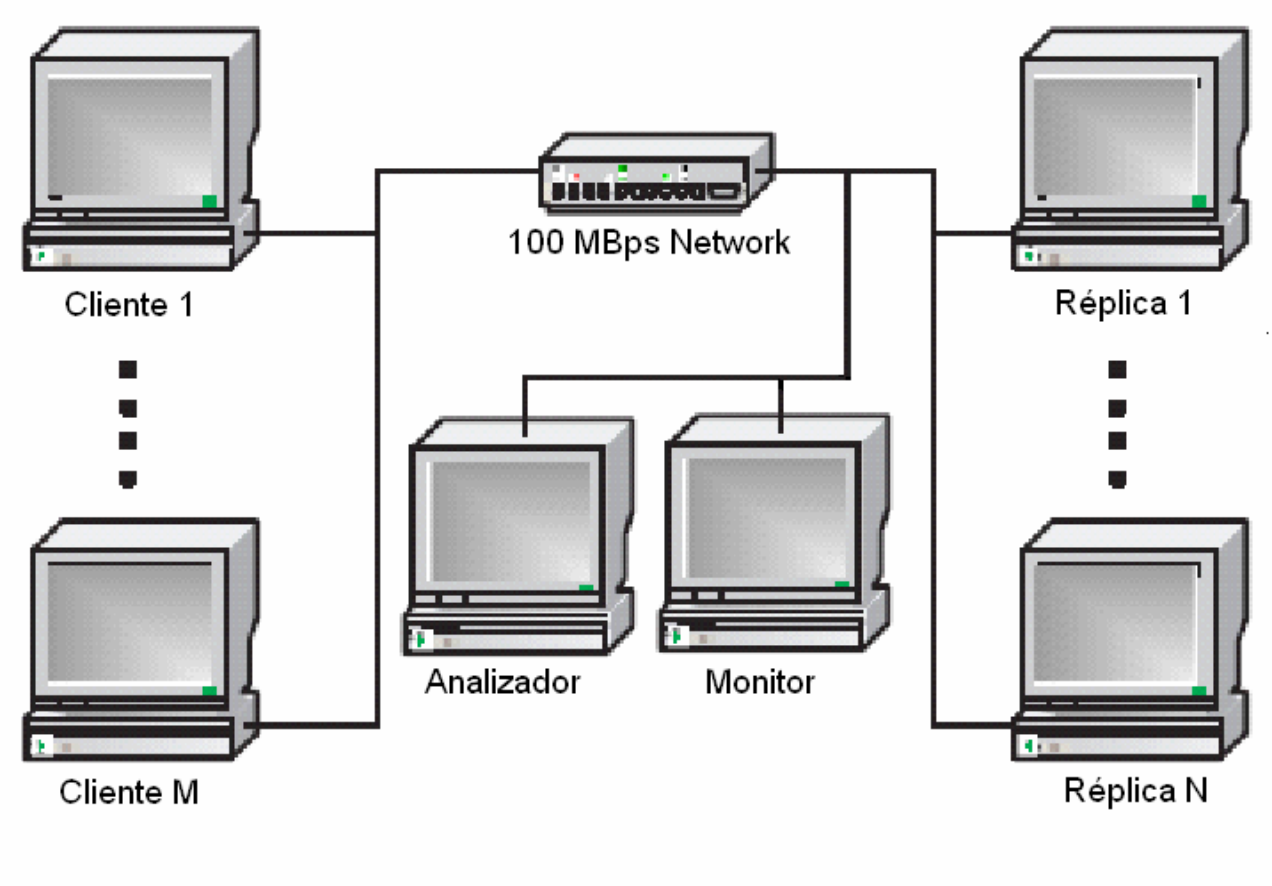


Figura 5.6 Arquitectura Heterogénea Física del Sistema.

5.4 Conclusiones.

El modelado es la parte central de todas las actividades que conducen a la producción de buen software. Construimos modelos para comunicar la estructura deseada y el comportamiento de nuestro sistema. Construimos modelos para visualizar y controlar la arquitectura del sistema. Construimos

modelos para comprender mejor el sistema que estamos construyendo. Construimos modelos para controlar el riesgo.

Por otra parte la programación orientada a objetos ofrece modularidad, encapsulación e identidad. Los objetos pueden ser manipulados y diseñados en una computadora de la misma manera como se representan en el mundo real. Además la flexibilidad del software orientado a objetos permite fácilmente la creación de aplicaciones prototipo, construcciones y cambios en respuesta a sus requerimientos.

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

El propósito del análisis fue proporcionar un modelo de la forma en que se comporta el sistema. Al enfocarnos sobre tal comportamiento identificamos los puntos claves que representan alguna actividad primaria en el sistema o con frecuencia denotan el mapeo de entradas y salidas que representan las transformaciones que el sistema hace en su ambiente.

El propósito del diseño fue crear una arquitectura para implementar el comportamiento que requiere el modelo del análisis. En este sentido, análisis y diseño unifican, usuarios y desarrolladores de un sistema en un vocabulario común para tratar el dominio del problema.

Capítulo 6. Análisis Comparativo de Estrategias de Balanceo Dinámico de Carga Bajo CORBA

La mejor forma de resolver un problema de balanceo de carga es centrar el problema como un problema de optimización [Goscinski 92]. 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 [Goldberg 89]. En este capítulo mostramos los resultados que arrojó un comparativo de cuatro estrategias de balanceo dinámico de carga versus nuestra estrategia Genética, las estrategias se implementaron bajo CORBA. Implementamos las tres estrategias propuestas por [IONA 02]: Round Robin, Random (Aleatoria) y Cola Más Corta "Least_Loaded" (CMC). Adaptamos una estrategia Umbral de los algoritmos emisores según Eager, Lazowska y Zohorjan [Eager 86] y por último diseñamos nuestra estrategia Genética. El comparativo se evaluó utilizando diferentes métricas de carga y se ilustra cómo la estrategia Genética bajo ciertas condiciones del sistema arrojó mejores resultados. Este trabajo se fundamentó principalmente en Othman Ossama and Douglas C. Schmidt en [Ossama 01a, 01b, 01c, 03a, 2003b], IONA Technologies [IONA 02], A., Hisao Kameda [Kameda 97], Grady Booch [Booch 94, 99], Behrooz A. Shirazi [Shirazi 95], y Randy L. Haupt [Haupt 98].

6.1 Introducción a los Algoritmos Genéticos.

Un algoritmo genético, es un algoritmo matemático altamente paralelo que transforma un conjunto (población) de objetos matemáticos individuales (típicamente cadenas de caracteres de longitud fija que se ajustan al modelo de las cadenas de cromosomas), cada uno de los cuales se asocia con una aptitud, en una población nueva (es decir, la siguiente generación) usando operaciones modeladas de acuerdo al principio Darwiniano de reproducción y sobrevivencia del más apto y tras haberse presentado de forma natural una serie de operaciones genéticas (notablemente la recombinación sexual)." [Koza 92].

En la Figura 6.1 mostramos un algoritmo genético, este empieza con un conjunto de varios individuos y aplica operadores de selección, cruzamiento y mutación para que evolucione un individuo satisfactoriamente medido por una función de adaptación.

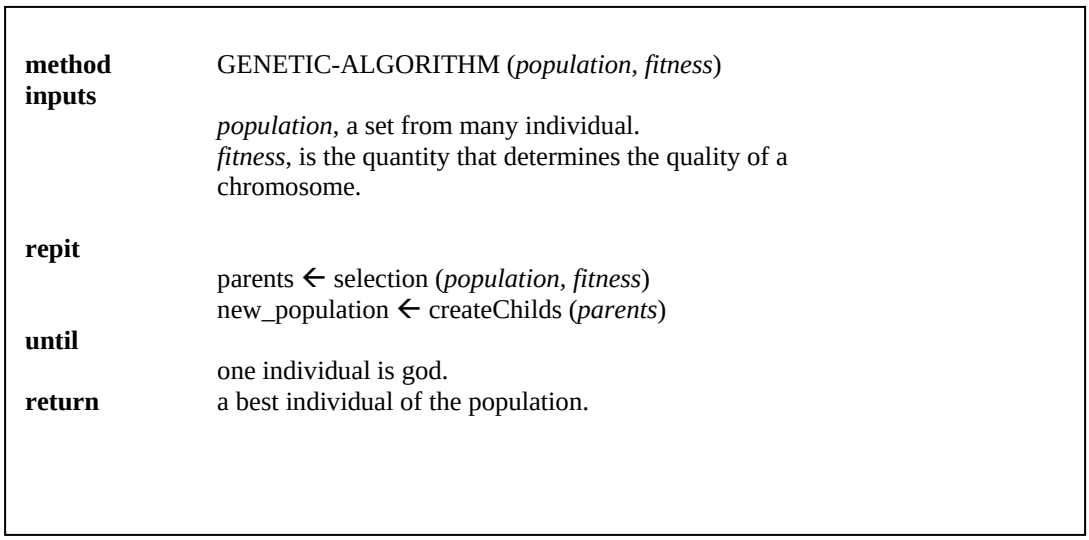


Figura 6.1. Algoritmo Genético con Evolución Simulada [Russell 95].

En la siguiente sección se muestra el teorema del esquema con el propósito de dar fundamento a los algoritmos genéticos, posteriormente describimos como implementamos los AGs en nuestro problema visualizando: la longitud de los genes, composición del cromosoma, población inicial y proceso evolutivo. La teoría de los AGs y los conceptos fundamentales pueden encontrarse más detalladamente en el apéndice A de este escrito.

6.2 Teorema del Esquema.

Mientras un AG explícitamente procesa cadenas, implícitamente este procesa esquemas, los cuales representan similaridades entre las cadenas. El AG no puede proponerse visitar todos los puntos en el espacio de búsqueda, y sí muestrear un número suficientemente grande de hiperplanos en regiones de alta adaptabilidad del espacio de búsqueda. Cada hiperplano corresponde a un conjunto de subcadenas similares de alta adaptabilidad.

Definición - Un *esquema* es una cadena de longitud l (la longitud de las cadenas de la población), tomada del alfabeto {0,1,*} donde * permite parear con 0 ó 1 en una posición particular. El **"*"** es un meta símbolo, que es una herramienta de notación para describir todas las posibles similaridades entre las cadenas.

Cada esquema representa el conjunto de todas las cadenas de longitud l, en cuyas posiciones con

valores 0 y 1 son idénticas.

Ejemplo:
El esquema 10**1, representa el conjunto {10001,10011,10101,10111}. Los elementos de este conjunto se conocen como instancias del esquema que ellos representan. Los esquemas son llamados también, *subconjuntos de similaridad* porque ellos representan subconjuntos de cadenas con similaridades en ciertas posiciones fijas. Las posiciones fijas de un esquema son las posiciones de la cadena con valores 0 y 1.

Sin embargo, no todos los esquemas son creados igual, algunos son más específicos que otros.

Ejemplo:
El esquema 0 1 1 * 1 * * es una construcción más definida acerca de las similaridades importantes que el esquema 0 * * * * * *. Además hay esquemas que se expanden más que otros.

Ejemplo:
El esquema 1 * * * * 1 * expande una porción más grande que el esquema 1 * 1 * * * * .
Dos propiedades importantes de los esquemas el *orden* y la *longitud definida* se relacionan con este hecho.

Definición - El *orden* de un esquema H, denotado por $o(H)$, es simplemente el número de posiciones fijas (es decir, de posiciones con valores en el alfabeto V = {0,1}) presentes en el patrón.

Ejemplo:
O(0 1 1 * 1 * *) = 4 y O(0 * * * * * *) = 1

Definición - La *longitud definida* de un esquema H, denotada por $\delta(H)$, es la distancia entre la primera y la última posición especificada de la cadena.

Ejemplo:

$\delta(0\ 1\ 1\ *\ 1\ *\ *) = 5 - 1 = 4$ y $\delta(0\ *\ *\ *\ *\ *\ *) = 0$

Dos conceptos se hacen necesarios para valorar la importancia de un esquema dentro de una población.

Definición **-** La *adaptabilidad promedio del esquema H*, denotada por $f(H)$, será definida como:

$$f(H) = \sum_{k=i1}^{im} f(k)/m \qquad (6.1)$$

donde $k = i_1,..., i_m$ son las cadenas instancias del esquema *H* y m la cantidad de dichas cadenas.

Definición .- La *adaptabilidad promedio de la población total*, denotada por ftotal, será:

$$ftotal = \sum_{j=1}^{n} f(j)/n \qquad (6.2)$$

j = 1 ,..., n, n el tamaño de la población.

Con esta información es posible hablar del concepto de *Bloques Constructores.*

Definición -Se denominan *Bloques Constructores* a los esquemas que son de longitud definida pequeña, de bajo orden y de adaptabilidad promedio por encima de la población total.

Sobre estos Bloques Constructores es que se define el incremento o decremento de los esquemas en una población y se analiza el efecto combinado de la selección, cruzamiento y mutación. Esto se conoce como ***Teorema de Esquema o Teorema Fundamental de los AG*** como lo califica Goldberg en su libro [Goldberg 89].

6.2.1 Teorema del Esquema (Efecto de Supervivencia).

Sea P_{cruce}, la probabilidad de que un par de cromosomas sea entrecruzado y sea P_{mut}, la probabilidad de que una mutación ocurra en un locus dado. Si $m(H,t)$, es el número de instancias del esquema H en la población en el tiempo t, entonces:

$$m(H,t+1) \geq m(H,t) * \frac{f(H)}{\bar{f}} \left[1 - P_{cruce} \frac{\delta(H)}{l-1} - o(H) P_{mut} \right] \quad (6.3)$$

Demostración:

Para demostrar el teorema del esquema, Holland analiza los efectos de la selección, cruzamiento y mutación como sigue [Goldberg 89]:

6.2.2 Efecto de la Selección Sobre el Número Esperado de Instancias de un Esquema en la Población t+1.

Suponemos que en un tiempo dado t hay m ejemplos de un esquema particular H contenido con la población $A(t)$ donde escribimos $m = m(H,t)$. Durante la reproducción, una cadena es copiada de acuerdo a su función de adaptabilidad, o más precisamente una cadena A_i es seleccionada con probabilidad:

$$p_i = f_i / \sum f_j \quad (6.4)$$

Después de seleccionada una población no sobrepuesta de tamaño n con remplazamiento, de la población $A(t)$, esperamos tener:

$$m(H,t+1) = m(H,t) * n * f(H) / \sum f_j \quad (6.5)$$

donde, $f(H)$ es la función de adaptación promedio de la representación de cadenas del esquema H en un tiempo t. Si reconocemos que la función de adaptación promedio de la población entera debe ser escrita como $\bar{f} = \sum f_j / n$ entonces debemos de re-escribir el esquema reproductivo como:

$$m(H, t+1) = m(H, t)\frac{f(H)}{\bar{f}} \tag{6.6}$$

En otras palabras un esquema particular crece a razón de la adaptabilidad promedio del esquema sobre la adaptabilidad promedio de la población. Por otra parte, esquemas con valores promedios sobre el promedio de la población recibirán un número incremental de ejemplos en la próxima generación, mientras esquemas con valores de función de adaptación por debajo del promedio de la población recibirá un decremento. Este comportamiento es llevado a cabo en todo el esquema H contenido en una población particular A en paralelo. En otras palabras, todos los esquemas en la población crecen o decaen de acuerdo al promedio de sus esquemas en la operación de reproducción.

Los efectos de reproducción sobre un número de esquemas son cualitativamente claros, esquemas por arriba del promedio crecen y esquemas por debajo del promedio mueren. ¿Podemos aprender una cosa más acerca de la forma matemática de crecimiento (o decadencia) de la ecuación de esquemas diferentes? Se asume que un esquema particular H rebasa al promedio con una cantidad $c\bar{f}$ con c como una constante. Dentro de esta suposición podemos re-escribir la ecuación del esquema como sigue:

$$m(H, t+1) = m(H, t)\frac{\left(\bar{f} + c\bar{f}\right)}{\bar{f}} = (1+c) * m(H, t) \tag{6.7}$$

Comenzando en $t = 0$ y asumiendo un valor estacionario de c, obtenemos la ecuación.

$$m(H, t) = m(H, 0) * (1+c)^t \tag{6.8}$$

Las personas que estén asociados con los negocios reconocerán que esta ecuación es la ecuación de interés compuesto, y las personas que estén asociadas con matemáticas reconocerán una progresión geométrica o una forma exponencial analógica discreta. Los efectos de reproducción son ahora cualitativamente claros; la reproducción aloja un incremento exponencial sobre los esquemas que están por arriba del promedio de la población y un decremento exponencial sobre los esquemas que están por debajo del promedio de la población.

6.2.3 Efecto del Cruzamiento Sobre el Número Esperado de Instancias de un Esquema en la Población t+1.

Para algunos la extensión de esto es curioso como la reproducción puede alojar un incremento o un decremento exponencial de esquemas en paralelo en futuras generaciones, muchos esquemas diferentes son ejemplificados en paralelo acorde a la misma regla a través del uso de n operaciones de reproducción. Por otra parte la reproducción sola no hace nada en la promesa de nuevas regiones del espacio de búsqueda, dado que ningún nuevo punto es buscado; si nosotros solamente copiamos estructuras viejas sin ningún cambio, entonces ¿como intentaremos algo nuevo?. Aquí es donde el paso de cruzamiento comienza. El cruzamiento es una estructura que produce información aleatoria intercambiada entre cadenas. El cruzamiento crea nuevas estructuras con un mínimo de ruptura para alojar estrategias dictadas por la sola reproducción. Estos resultados se incrementan (o decrementan) exponencialmente en porciones del esquema en una población de muchos esquemas contenidos en la población.

¿Cuáles esquemas son afectados por cruzamiento? y ¿cuáles no?. Considere una cadena particular de longitud $l = 7$ y dos esquemas representativos con esa cadena.

Ejemplo: Sea $l = 7$,
A = 0 1 1 1 0 0 0
H_1 = * 1 * * * * 0
H_2 = * * * 1 0 * *

Los dos esquemas H_1 y H_2 son representados en la cadena A. Se utiliza un cruzamiento simple como el

descrito en la sección anterior. Al determinar el punto de cruce, éste produjo la ruptura de las cadenas entre las posiciones 3 y 4, como se observa en el siguiente ejemplo.

Ejemplo:

A = 0 1 1 | 1 0 0 0

H_1 = * 1 * | * * * 0

H_2 = * * * | 1 0 * *

El esquema H_1 será destruido porque el 1 de la posición 2 y el 0 de la posición 7 serán ubicados en diferentes descendientes (ellos están en lados opuestos del separador que marca el punto de cruce). Con el mismo punto de cruce H_2 sobrevivirá porque el 1 de la posición 4 y el 0 de la posición 5 pasan intactos a un único descendiente. El esquema H_1 es menos probable que sobreviva a un cruzamiento que el esquema H_2.

Si el sitio del cruzamiento es seleccionado de forma aleatoriamente uniforme entre $l-1$ = 7-1 = 6 sitios posibles, entonces el esquema H_1 es destruido con probabilidad

$$p_{destrucción} = \delta(H_1)/(l-1) = 5/6 \tag{6.9}$$

éste sobrevive con probabilidad,

$$p_{\sup ervivencia} = 1 - p_{destrucción} = 1/6 \tag{6.10}$$

Para el esquema H_2 la longitud es $\delta(H_2) = 1$ y este es destruido durante un evento en seis cuando el sitio de corte sea seleccionado entre las posiciones 4 y 5, así pues $p_{destrucción} = 1/6$ o la probabilidad de supervivencia es $p_{\sup ervivencia} = 1 - p_{destrucción} = 5/6$.

Más generalmente, un límite inferior sobre la probabilidad de supervivencia al cruzamiento $p_{\sup ervivencia}$ puede ser calculada para cualquier esquema. Como, un esquema sobrevive, cuando el sitio de cruzamiento cae fuera de la longitud definida, la probabilidad de supervivencia bajo un simple

cruzamiento es:

$$p_{\sup ervivencia} = 1 - \frac{\delta(H)}{(l-1)} \tag{6.11}$$

ya que el esquema puede ser destruido si el sitio de cruzamiento cae entre las posiciones de los valores bien definidos del esquema que determina su longitud definida y hay $l-1$ posibles formas de destruirlo. Luego, la posibilidad de destruirlo es $\delta(H)/(l-1)$.

Si el cruzamiento es el mismo realizado por selección aleatoria digamos con probabilidad p_{cruce} en un pareo particular, la probabilidad de supervivencia puede darse por la expresión.

$$p_{\sup ervivencia} \geq 1 - p_{cruce} * \frac{\delta(H)}{l-1} \tag{6.12}$$

la cual se reduce a la primera expresión cuando p_{cruce}=1.0.

El efecto combinado de la selección y cruzamiento puede considerarse en la siguiente expresión:

$$m(H, t+1) = m(H, t) * \frac{f(H)}{\bar{f}} \left[1 - p_{cruce} * \frac{\delta(H)}{l-1} \right] \tag{6.13}$$

El esquema H crece o muere dependiendo de un factor de multiplicación. Ahora, este factor depende de si el esquema está por encima o por debajo del promedio de la población y si el esquema tiene longitud definida relativamente pequeña o grande. *Claramente, aquellos esquemas con promedio alto y longitud definida pequeña, serán muestreados con una razón de incremento exponencial.*

6.2.4 Efecto de la Mutación sobre el Número de Instancias de un Esquema en la Población t+1.

Usando nuestra definición previa, mutación es la alteración aleatoria de una simple posición con probabilidad p_m. De la manera en que un esquema H sobreviva, es que todas las posiciones fijas

necesitan ellas mismas sobrevivir. Por tanto, dado que un simple alelo sobrevivirá con probabilidad $(1-p_m)$, y dado que cada una de sus mutaciones es estadísticamente independiente, un esquema particular sobrevivirá cuando cada una de las $o(H)$ posiciones fijas en el esquema sobre-vivan. Multiplicando la probabilidad de supervivencia $(1-p_m)$ por $o(H)$ veces, obtenemos la probabilidad de supervivencia de la mutación $(1-p_m)^{o(H)}$**.** Para valores pequeños de $p_m (p_m << 1)$, la probabilidad de supervivencia del esquema puede aproximarse por la expresión:

$$1-o(H)*p_m \tag{6.14}$$

Por tanto se llega a que un esquema particular H recibe un número esperado de instancias en la próxima generación bajo selección, cruzamiento y mutación, dada por la siguiente ecuación (ignorando pequeños términos del producto cruzado):

$$m(H,t+1) = m(H,t)*\underbrace{\frac{f(H)}{\bar{f}}}_{\text{Selección}}\underbrace{\left[1-p_{cruce}*\frac{\delta(H)}{l-1}\right]}_{\text{Cruzamiento}}\underbrace{*o(H)p_m}_{\text{Mutación}} \tag{6.15}$$

Nuestra conclusión sería: *esquemas arriba del promedio, cortos y de bajo orden reciben un incremento exponencial en las siguientes generaciones*. Esta conclusión es tan importante que damos a esta un nombre especial: *El Teorema del Esquema o Teorema Fundamental de los Algoritmos Genéticos.*

6.2.5 Como Trabaja el Procesamiento de Esquemas: Un Ejemplo Revisado Manualmente.

Observemos tres esquemas particulares, que llamamos H_1 , H_2 y H_3, donde:

$H_1 = 1****$

$H_2 = *10**$

$H_3 = 1***0$

PROCESAMIENTO DE CADENAS

Cadena	Población Inicial	Valor x	f(x) x∧2	Pselección $f_i/\Sigma f$	Count Esperado fi/promedio(f)	Count Actual (Ruleta)
1	01101	13	169	.14	.58	1
2	11000	24	576	.49	1.97	2
3	01000	8	64	.06	0.22	0
4	10011	19	361	.031	1.23	1
	Sum =>		1170	1.00	4.00	4.0
	Promedio =>		293	0.25	1.00	1.0
	Max =>		576	1.23	1.97	2.0

PROCESAMIENTO DE ESQUEMAS

		Cadenas Representativas	Fitness Promedio del Esquema
H1	1****	2,4	469
H2	*10**	2,3	320
H3	1***0	2	576

Tabla 6.1 a) Procesamiento de Esquemas.

Observe los efectos de reproducción, cruzamiento y mutación sobre el primer esquema H_1. Durante la fase de reproducción, la cadena es copiada probabilísticamente de acuerdo a sus valores de adaptación (fitness). Miramos la primer columna de la tabla 6.1(a), notamos que las cadenas 2 y 4 son ambas representativas del esquema 1****. Después de la reproducción, notamos que tres copias del esquema han sido producidas (cadenas 2, 3 y 4 en la columna de reproducción). ¿Este número corresponde con el valor predefinido en el teorema del esquema?. Del teorema del esquema esperamos tener $m * f(H)/\bar{f}$ copias. Calculando el promedio del esquema $f(H_1)$, obtenemos (576 + 361)/2 = 468.5. Dividiendo este valor entre el valor promedio de la población $\bar{f} = 293$ y multiplicando por el número de esquemas H_1 en tiempo t, $m(H_1, t) = 2$, obtenemos el número esperado de esquemas H_1 en un tiempo $t+1$, $m(H, t+1) = 2 * 468.5 / 293 = 3.19$. Comparando este con el número actual de esquemas (tres), vemos que obtenemos un número correcto de copias. Tomando esto en un paso posterior,

observamos que un cruzamiento no puede tener algún efecto posterior sobre este esquema por que una definición de longitud $\delta(H_1) = 0$ previene la ruptura de un simple bit. Posteriormente, con la razón de mutación a $p_m = 0.001$ esperamos tener $m * p_m = 3 * 0.001 = 0.003$ o ningún bit cambiado con las tres copias del esquema en las tres cadenas. Como un resultado, observamos que para el esquema H_1, obtenemos el incremento exponencial de esquemas como se predijo en el teorema del esquema.

Esto es algo grande y bueno, pero el teorema del esquema con un solo bit predefinido, parece ser un caso especial. ¿Que hay acerca de la propagación de similitudes importantes con grandes longitudes definidas?. Por ejemplo considere la propagación del esquema $H_2 = *10**$ y del esquema $H_3 = 1***0$. Siguiendo la reproducción y el cruzamiento, la replicación del esquema es correcto. En el caso del esquema H_2 comienza con dos ejemplos de la población inicial y termina con dos copias siguiendo la reproducción. Esto coincide con el número esperado de copias, $m(H_2) = 2 * 320 / 293 = 2.18$, donde 320 es el promedio del esquema y 293 es el promedio de la población. En el caso de H_3 comienza con un simple ejemplo (cadena 2) y termina con dos copias siguiendo la reproducción (cadena 2 y 3 en la columna de copias de cadenas). Este acuerdo con el número esperado de copias $m(H3) = 1 * 576 / 293 = 1.97$, donde 576 es el promedio de los esquemas y 293 es el promedio de la población. Note que para un esquema corto, el esquema H_2, las dos copias son mantenidas aún después del cruzamiento. Porque al definir la longitud corta sólo interrumpimos el proceso una sola vez en cuatro $(l - 1 = 5 - 1 = 4)$. Como un resultado, el esquema H_2 sobrevive con alta probabilidad. El número esperado de esquemas H_2 es $m(H_2, t+1) = 2.18 * 0.75 = 1.64$ y este se compara bien con el conteo actual de dos esquemas. H_3, es un esquema de diferente color. Porque define la longitud con $(\delta(H_3) = 4)$, el cruzamiento usualmente destruye este esquema.

PROCESAMIENTO DE CADENAS

Mating Pool Después de la Reproducción	Mate	Punto de Cruce	Nueva Población	Valor X	f(x) x∧2
0110|1	2	4	01100	12	144
1100|0	1	4	11001	25	625
11|000	4	2	11011	27	729
10|011	3	2	10000	16	256
Sum =>					1754
Promedio =>					439
Max =>					729

PROCESAMIENTO DE ESQUEMAS

Después de la Reproducción			Después de Todos los Operadores		
Count Esperado	Count Actual	Cadenas Representativas	Count Esperado	Count Actual	Cadenas Representativas
3.20	3	2,3,4	3.20	3	2,3,4
2.18	2	2,3	1.64	2	2,3
1.97	2	2,3	0.0	1	4

Tabla 6.1 b) Procesamiento de Esquemas.

Estos cálculos han confirmado la teoría desarrollada anteriormente. Esquemas cortos, y de bajo orden se dan en un incremento o decremento exponencial dependiendo de la función de adaptación promedio de los esquemas.

Ejemplos:

1) Considerando tres cadenas A_1 = 11101111, A_2 = 00010100 y A_3 = 01000011 y seis esquemas H_1 = 1*******, H_2 = 0*******, H_3 = ******11, H_4 = ***0*00*, H_5 = 1*****1* y H_6 = 1110**1*.

Y la pregunta es ¿Cuáles esquemas son emparejados con cuales cadenas? ¿Cuál es el orden y longitud de cada esquema? Estimar la probabilidad de supervivencia de cada esquema bajo mutación, cuando la probabilidad de una mutación simple es $p_m = 0.001$. Estima la probabilidad de supervivencia de cada esquema bajo cruzamiento cuando la probabilidad de cruzamiento es $p_c = 0.85$.

Cadena	Correspondencia	O(H)	δ(H)	P_s (P_m=.001)	P_s (P_c=.85)
A_1=11101111					
A_2=00010100					
A_3=01000011					
H_1=1*******	(A_1)	1	0	0.999	1
H_2=0*******	(A_2,A_3)	1	0	0.999	1
H_3=******11	(A_1,A_3)	2	1	0.998	0.8785
H_4=***0*00*	ninguno	3	3	0.997	0.6357
H_5=1*****1*	(A_1)	2	6	0.998	0.2714
H_6=1110**1*	(A_1)	5	6	0.995	0.2714

Tabla 6.2 Ejemplos para Procesar Esquemas.

2) Una población contiene las siguientes cadenas y valores de aptitud en la generación 0:

#	Cadena	Aptitud
1	10001	20
2	11100	10
3	00011	5
4	01110	15

Tabla 6.3 Cadenas y Valores de Aptitudes.

La probabilidad de mutación es $P_m = 0.01$ y la probabilidad de cruzamiento es $p_c = 1.0$. Calcula el número esperado de esquemas de la forma 1**** en la generación 1. Estima el número esperado de esquemas de la forma 0**1* en la generación 1.

#	Cadena	Aptitud	Pselección $f_i/\Sigma f$	Count Esperado $fi/\mu(f)$	Count Actual
1	10001	20	.4	1.6	2
2	11100	10	.2	.8	1
3	00011	5	.1	.4	0
4	01110	15	.3	1.2	1
	Sum =>	50	1	4	4
	Promedio =>	12.5	0.25	1	1
	Max =>	20	0.4	1.6	2

Tabla 6.4 Teorema del Esquema $m = m(H, t+1)$.

H_1 = 1**** representa a las cadenas (1,2) y antes de reproducción f(H) = 15

H_2 = 0**1* representa a las cadenas (3,4) y antes de reproducción f(H) = 10

Y aplicando las fórmulas del teorema de esquema, encontramos que el número de esquemas esperado para la próxima generación será: H_1 = 2.376 y H_2 = 0.368, aumentando la probabilidad de éxito de estas cadenas en:

$$H_1 = 2 * 15 / 12.5 * (1 - 1* (0/4) - 2 * .01) = 2.376$$
$$H_2 = 2 * 10 / 12.5 * (1 - 1 *(3/4) - 2 * .01) = 0.368$$

6.3 Diseño de Nuestro Algoritmo Genético Bajo CORBA.

Una vez que hemos descrito la teoría que fundamentan los AGs, las siguientes secciones tienen como objetivo mostrar como implementamos un AG para el balanceo dinámico de carga en CORBA, para ello explicamos como se diseña la longitud de los genes, composición del cromosoma y población inicial. También explicamos el ciclo genético, implementando y desarrollando el proceso evolutivo y la manera en que se construye la función objetivo, pensando siempre en realizar tareas de búsqueda optimizada con el objetivo de tener una mejor planeación de carga que pueda mejorar el tiempo de respuesta global del sistema.

Número de Réplica	Representación Binaria (Genes)
1	000
2	001
3	010
4	011
5	100
6	101
7	110
8	111

Tabla 6.5 Representación de las Réplicas.

6.3.1 Longitud de los Genes.

[Bigus 01] dice que cada población tiene una gran cantidad de cromosomas, consistiendo de un gran número de genes generados por el proceso de selección natural, los cromosomas generalmente tienen una representación binaria, pero pudieran ser valores reales, estos cromosomas deben ser evaluados en forma aleatoria mediante algún proceso de cruzamiento para generar nuevos cromosomas, los cromosomas con mejores características sobreviven, en tanto que los cromosomas con pobres características son descartados. Nosotros generamos nuestra población inicial de manera aleatoria, cada cromosoma de la población esta compuesto de varios genes que son construidos en base al número de réplicas de la siguiente manera:

$$BPR = \frac{log(Number\_of\_Replicas)}{log(2)} \tag{6.16}$$

Donde:

BPR.. Número de Bits por Réplica.
Log(Number_of_Replicas). Logaritmo en Base al Número de Réplicas.
Log(2). Logaritmo Base 2.

La Tabla 6.5 muestra cada uno de los genes que representan las réplicas donde una o más demandas serán procesadas.

6.3.2 Composición del Cromosoma.

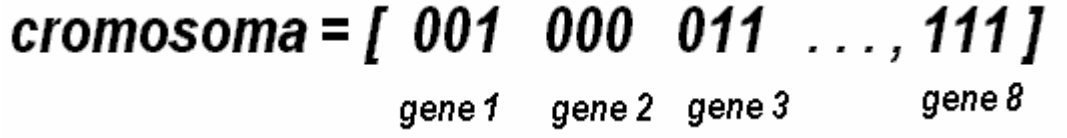
Los cromosomas, son arreglos de bits que son seleccionados de manera aleatoria para sobrevivir y procrear la próxima generación de cromosomas. Nuestros cromosomas están compuestos en función del número de demandas y el número de réplicas. Una vez calculado el tamaño del gen que representa cada una de las réplicas en donde se procesarán las demandas el siguiente paso es calcular la longitud del cromosoma que esta dado por la fórmula siguiente:

$$L\_Chromosome = BPR \cdot N\_Requirements \tag{6.17}$$

Donde:

L_Chromosome. Longitud del Cromosoma.
N_Requirements. Número de Requerimientos.

Así, por ejemplo, si se supone que se tienen 8 demandas y 8 réplicas posibles para procesar esas demandas, la longitud del cromosoma (L_Cromosoma) estará compuesto por el producto, que es el tamaño del gene (compuesto de 3 bits) multiplicado por el número de demandas (8), por lo tanto la longitud del cromosoma para este caso especifico será de 24 bits.



6.3.3 Generando la Población Inicial.

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

$$IPOP = L\_Chromosome \cdot Nipop \tag{6.18}$$

Donde:

L_Chromosome. Longitud del Cromosoma.
Nipop. Número de Cromosomas en la Población Inicial.
IPOP. Matriz Binaria de la Población.

Cada renglón en la matriz es un cromosoma, cada cromosoma nos indica la forma en que serán planeadas las demandas en las réplicas. En la Figura 6.2 podemos observar las columnas en la matriz, cada una representa un cromosoma en la población. Cada población consiste de cromosomas binarios (puntos rojos y azules) que significan ceros y unos.

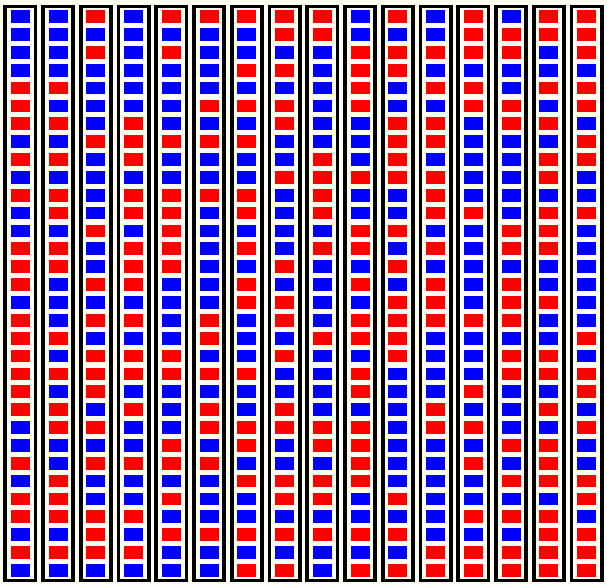


Figura 6.2 Población Binaria[Obitko 98].

6.3.4 Aplicando el Ciclo Genético.

6.3.4.1 El Proceso Evolutivo.

El proceso comienza por usar una técnica llamada elitismo. Cuando creamos una nueva población por cruzamiento y mutación, tenemos una gran oportunidad de perder el mejor cromosoma. Elitismo es el nombre del método que copia el mejor cromosoma a la nueva población [Bigus 01].

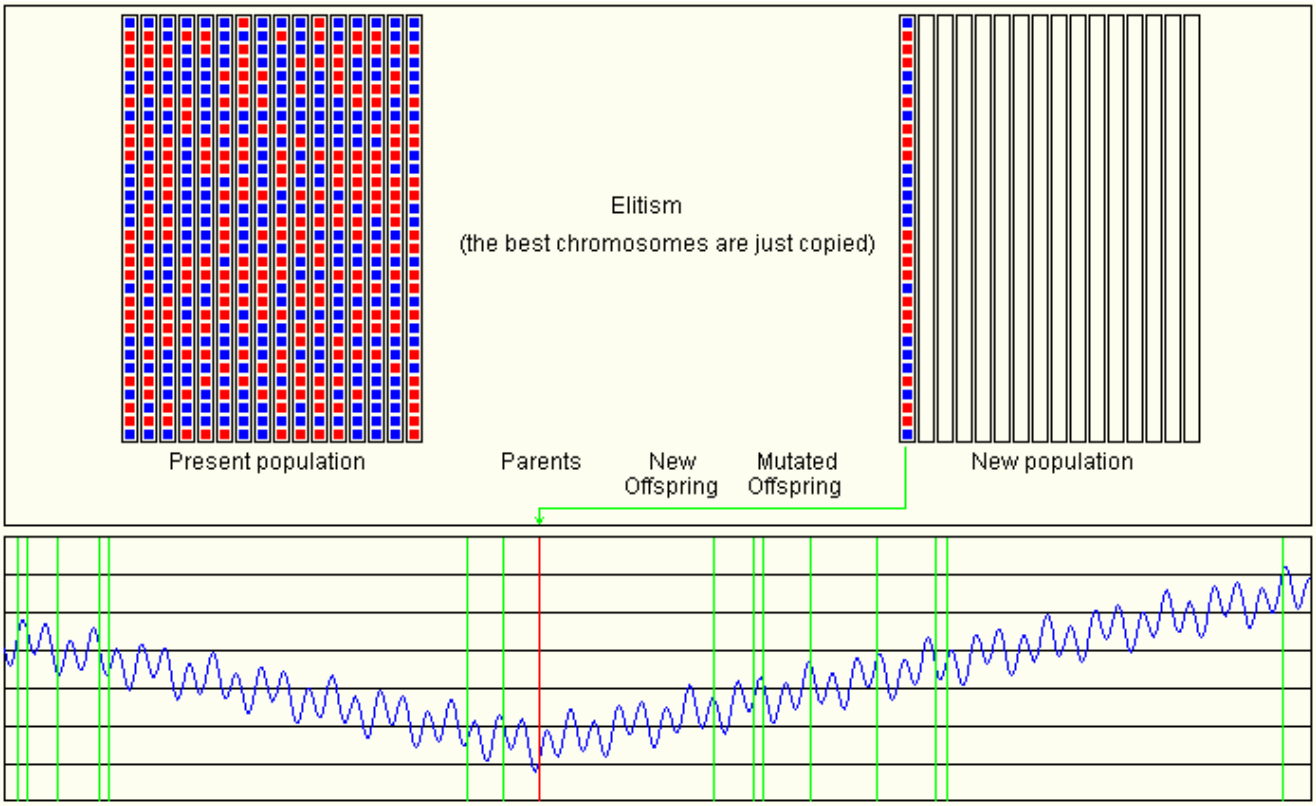


Figura 6.3 El mejor Miembro de la Población Actual es Copiado como el Primer Miembro de la Nueva Población[Obitko 98].

La gráfica de la Figura 6.3 representa el espacio de búsqueda de una población, las líneas verticales representan soluciones (puntos en el espacio de búsqueda). La línea roja representa la mejor solución, las líneas verdes representan otras soluciones, en la gráfica se observa claramente como el mejor miembro de la población (cromosoma) es copiado a la nueva población.

Después de elegir el mejor miembro de la población, dos padres son seleccionados aleatoriamente de la población aplicando el método de ruleta de la forma siguiente: [Goldberg 89, Bigus 01]

- Se calcula la suma de todos fitness de los cromosomas en la población (**sum S**).
- Se generan un número aleatorio (**-r**) entre 0 y la suma total de los fitness en la población.
- Se camina a través de la población. Cuando la suma (**sum S)** de los fitness sea mayor que **r** detenerse y retornar el cromosoma que se esté evaluando actualmente.

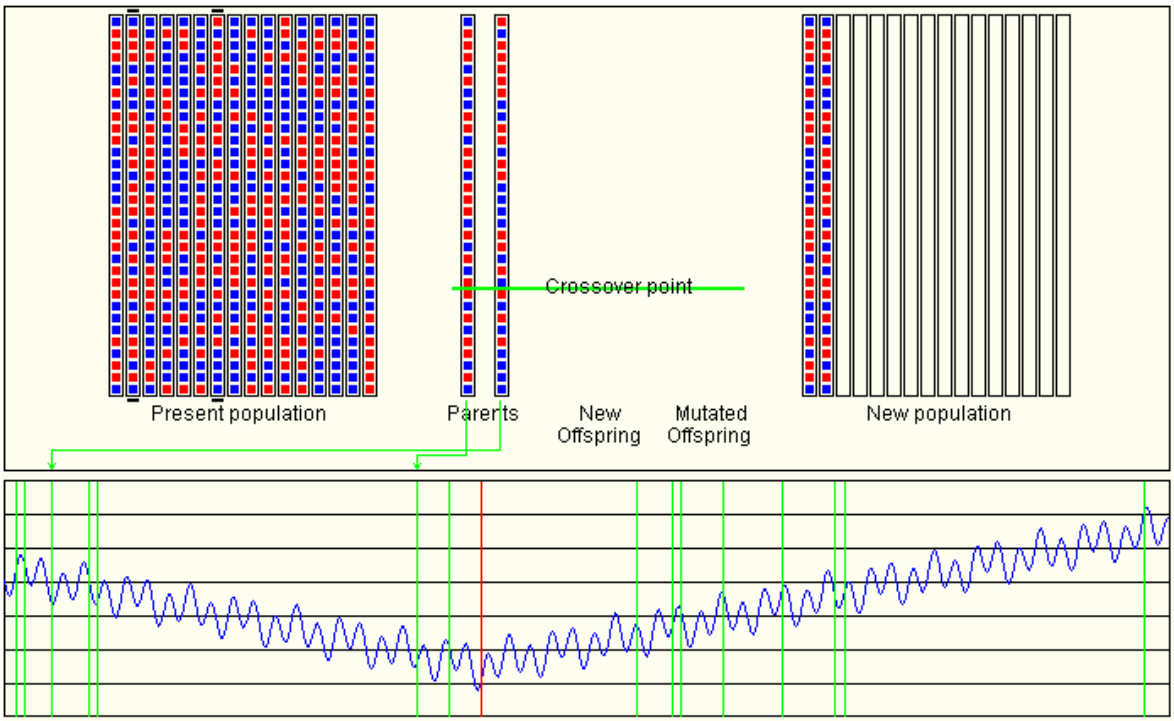


Figura 6.4 Operador de Cruce en un Punto [Obitko 98].

Una vez seleccionados los cromosomas padres uno de los siguientes operadores genéticos es seleccionado aleatoriamente.

1) Un operador de cruce en un punto.
2) Un operador de cruce en un punto junto con una mutación.
3) Sólo un proceso de mutación.

Operador de cruce en un punto. En la Figura 6.4 se puede observar el operador de cruce en un punto. Este operador toma el material genético de ambos padres, selecciona un punto de cruce y recombina el material genético de ambos padres. Primero se genera un número aleatorio entre 0 y 1. Si el número aleatorio es menor que una razón de cruce (0.85), la operación de cruzamiento toma lugar. Si el número aleatorio no es menor, los cromosomas padres se retornan como están. Si el cruzamiento se lleva a cabo, otro número aleatorio es generado y escalado a la longitud del cromosoma para definir la posición donde se hará la partición. Cada cromosoma padre es partido en dos sub-cadenas en el punto de cruce y son reensamblados con la sub-cadena izquierda del padre 1, combinada con la sub-cadena derecha del padre 2, y la sub-cadena derecha del padre 1 combinada con la sub-cadena izquierda del padre 2.

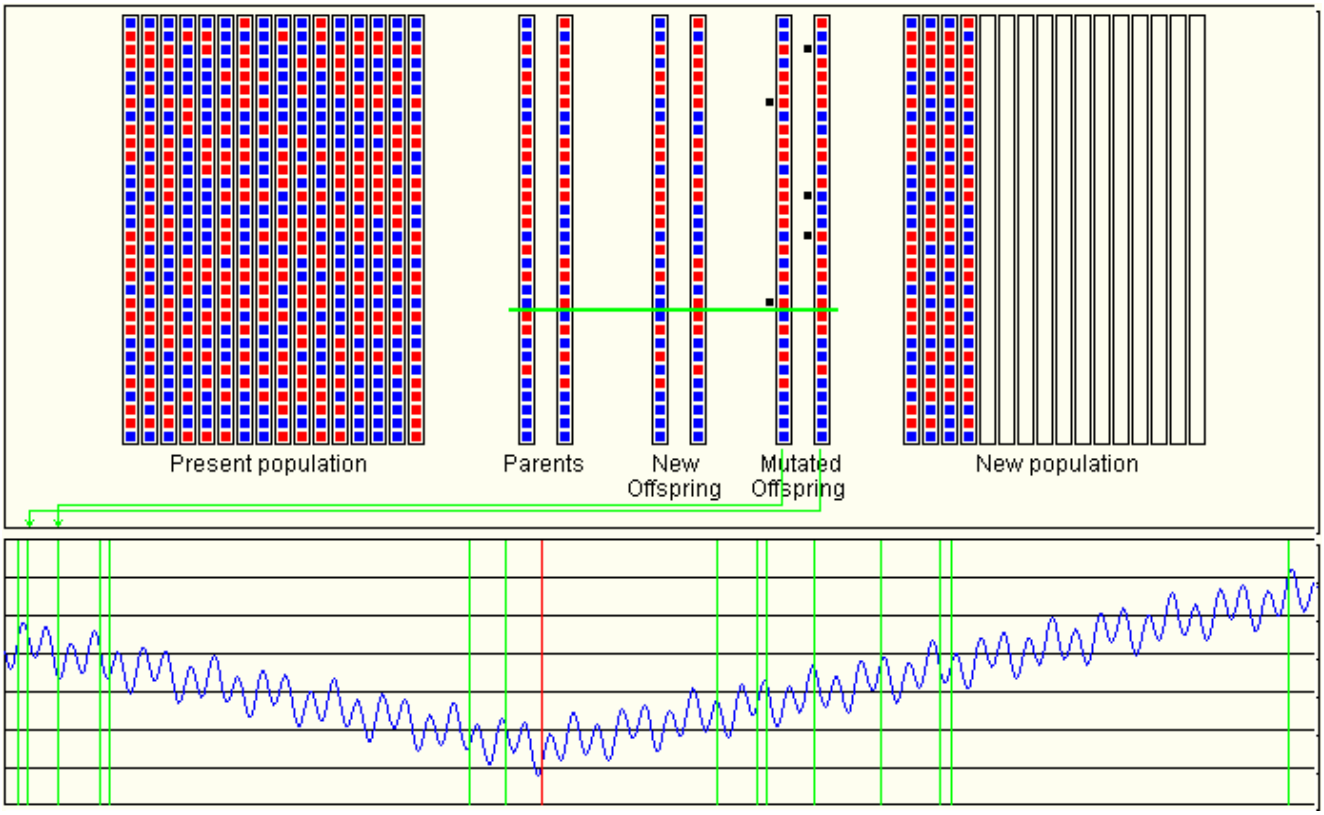


Figura 6.5 Operador de Cruce en un Punto Junto con una Mutación [Obitko 98].

Un operador de cruce en un punto junto con una mutación. Realiza las mismas operaciones que el operador de cruce en un punto y además ejecuta una *mutación* en cada cromosoma hijo, basándose en una razón de mutación (0.008). Un número aleatorio es generado entre 0 y 1. Si el número aleatorio es menor que la razón de mutación, entonces este bit es mutado sino otro número aleatorio es generado para obtener otro bit que será mutado. El proceso es repetido en un ciclo hasta alcanzar la longitud máxima del cromosoma (ver Figura 6.5).

Después de concluir el operador genético, los cromosomas hijos son retornados y serán usados para crear la nueva población.

6.3.4.2 Función Objetivo (Fitness).

La función objetivo de un algoritmo genético puede ser diseñada para ejecutar diferentes tareas de búsqueda de optimización. El valor de la función objetivo proporciona la cantidad para guiar el proceso de reproducción de los algoritmos genéticos a una nueva generación. Se construye la función objetivo tratando de minimizar la cola que tenga la mayor longitud de requerimientos dentro de las réplicas [Zomaya 01, Kidwell 94], éste valor es multiplicado por el promedio de las colas de las réplicas dividido entre la cola con longitud mayor de la forma siguiente:

$$fitness = Ccurrent_{(Lqs\_b)} \ldots + \frac{1}{max(Lqs\_b)} \cdot \frac{average(Lqs\_b)}{max(Lqs\_b)} \tag{6.19}$$

Donde:

fitness. Función Objetivo.
max(Lqs_b). Máxima Longitud de las Colas en las Réplicas.
average(Lqs_b). Promedio de las Colas de las Réplicas.
Ccurrent(Lqs_b). Carga en Ejecución de las Réplicas.

6.4 Diseñando Nuestras Estrategias.

Enseguida discutiremos un estudio comparativo de 5 estrategias de balanceo de carga que fueron implementadas en la arquitectura ya discutida en el capítulo 5. Cabe aclarar que en algunas ocasiones usamos algoritmo como sinónimo de estrategia, para nosotros ambas palabras tienen el mismo significado. Implementamos las tres estrategias propuestas por [IONA 02]: Round Robin, Random (Aleatoria) y Cola Más Corta “Least_Loaded” (CMC). Adaptamos una estrategia Umbral de los algoritmos emisores según Eager, Lazowska y Zohorjan [Eager 86] por último adaptamos nuestra estrategia Genética descrita previamente. El comparativo se avaluó utilizando diferentes métricas de carga y se ilustra cómo la estrategia Genética bajo ciertas condiciones del sistema arrojo mejores resultados.

6.4.1 Estrategia Aleatoria (Random).

La estrategia *aleatoria* es usada para balancear la carga en nuestra arquitectura. Esta estrategia es

no-adaptativa y selecciona aleatoriamente una réplica de un grupo de réplicas. El diagrama de secuencia mostrado en la Figura 6.6 describe lo siguiente:

1. El cliente solicita se de respuesta a sus requerimientos. Los requerimientos son interceptados transparentemente por el balanceador de carga.
2. El balanceador de carga elige la métrica y selecciona una réplica de manera aleatoria, posteriormente retorna la referencia de la réplica al cliente.
3. El cliente se comunica con la réplica para solicitarle que le de servicio a sus requerimientos.
4. En otro instante de tiempo, un nuevo cliente pide que se le de servicio a sus requerimientos.
5. Nuevamente el balanceador de carga selecciona la réplica de forma aleatoria. La réplica puede ser la misma o puede ser alguna otra del grupo dependiendo del valor aleatorio generado. El balanceador de carga una vez más retorna la referencia de la réplica al cliente.
6. El cliente se comunica nuevamente con la réplica enviando las peticiones de sus requerimientos.

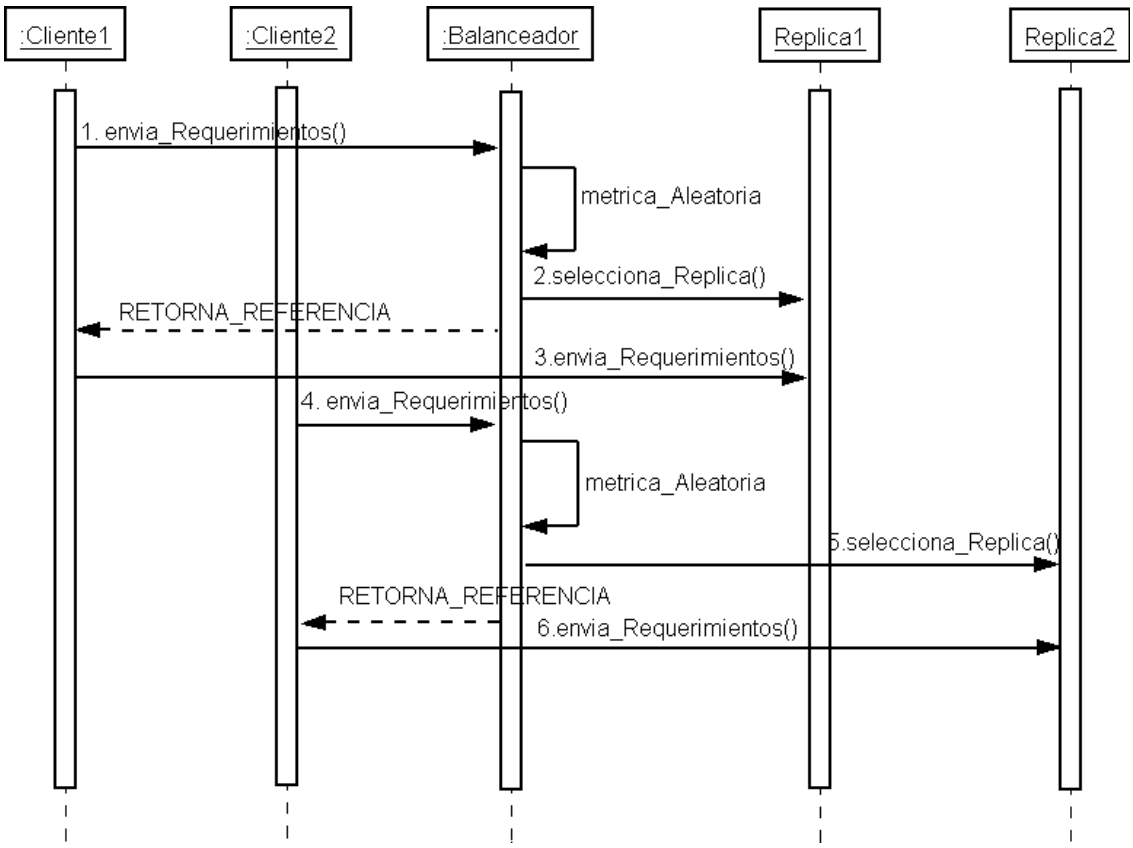


Figura 6.6 El Balanceador de Carga Selecciona una Réplica Aplicando la Estrategia Aleatoria.

6.4.2 Estrategia Round Robin.

Esta estrategia es una 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, esta estrategia selecciona rotativamente los miembros de un grupo de objetos dado. En otras palabras, todos los requerimientos de los clientes son igualmente distribuidos entre los servidores. El modelo funcional de esta estrategia es como sigue (ver Figura 6.7):

1. El cliente solicita se de respuesta a sus requerimientos.
2. Los requerimientos son interceptados transparentemente por el balanceador de carga. El balanceador de carga selecciona una réplica aplicando la estrategia Round Robin y retorna la referencia de la réplica al cliente.
3. El cliente se comunica con la réplica para solicitarle de servicio a sus requerimientos.
4. En otro instante de tiempo, un nuevo cliente pide que se le de servicio a sus requerimientos.
5. Nuevamente el balanceador de carga selecciona la réplica aplicando la estrategia Round Robin. La réplica es rotatoria en un grupo de réplicas. Una vez seleccionada la réplica el balanceador de carga retorna la referencia de la réplica al cliente.
6. El cliente se comunica nuevamente con la réplica enviando las peticiones de sus requerimientos.

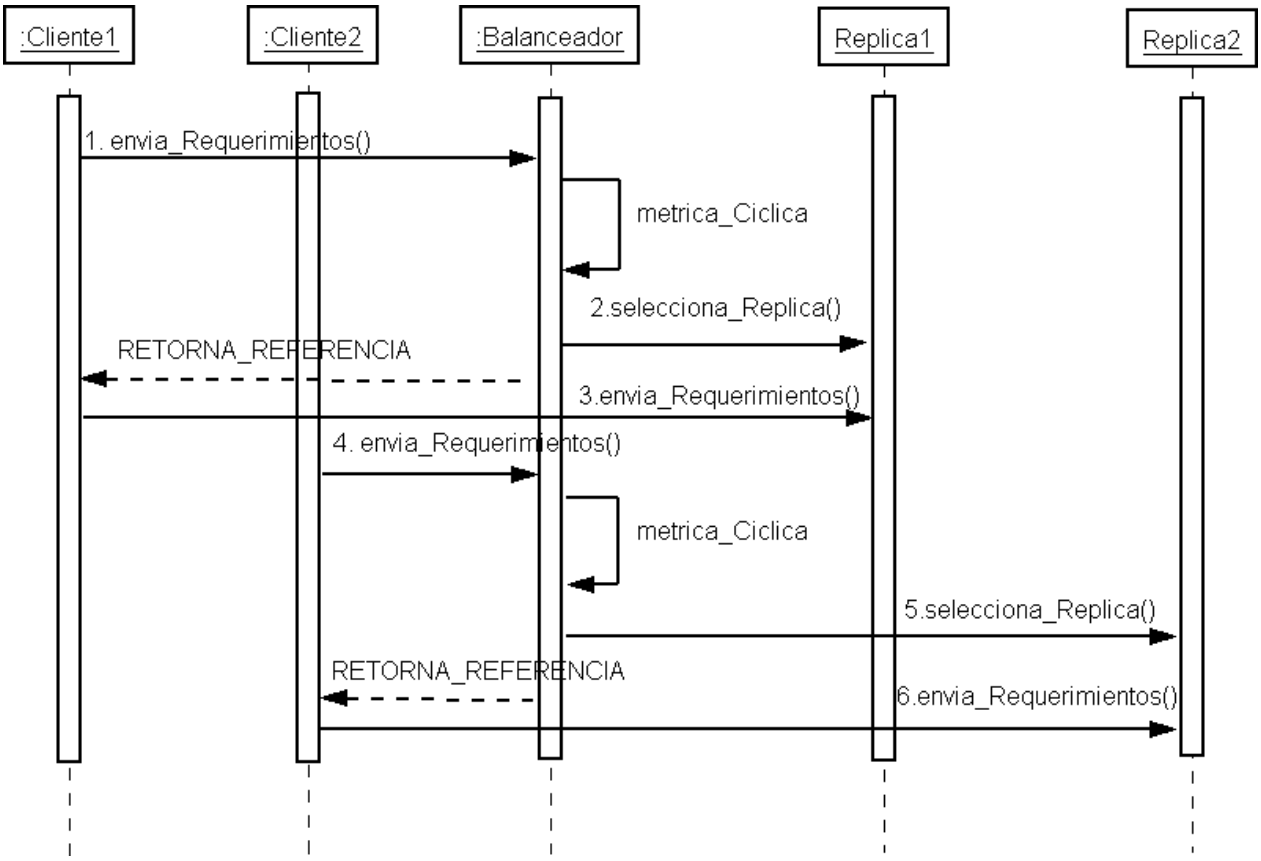


Figura 6.7 El Balanceador de Carga Selecciona una Réplica Aplicando la Estrategia Round Robin

6.4.3 Cola Más Corta (Least_Loaded).

La estrategia de la *Cola Más Corta* es utilizada para el balanceo dinámico de carga en nuestra arquitectura. A diferencia de la estrategia *Random* y *Round Robin*, la estrategia de la *Cola Más Corta* 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. El comportamiento de esta estrategia es el que se muestra en la Figura 6.8 y describe lo siguiente:

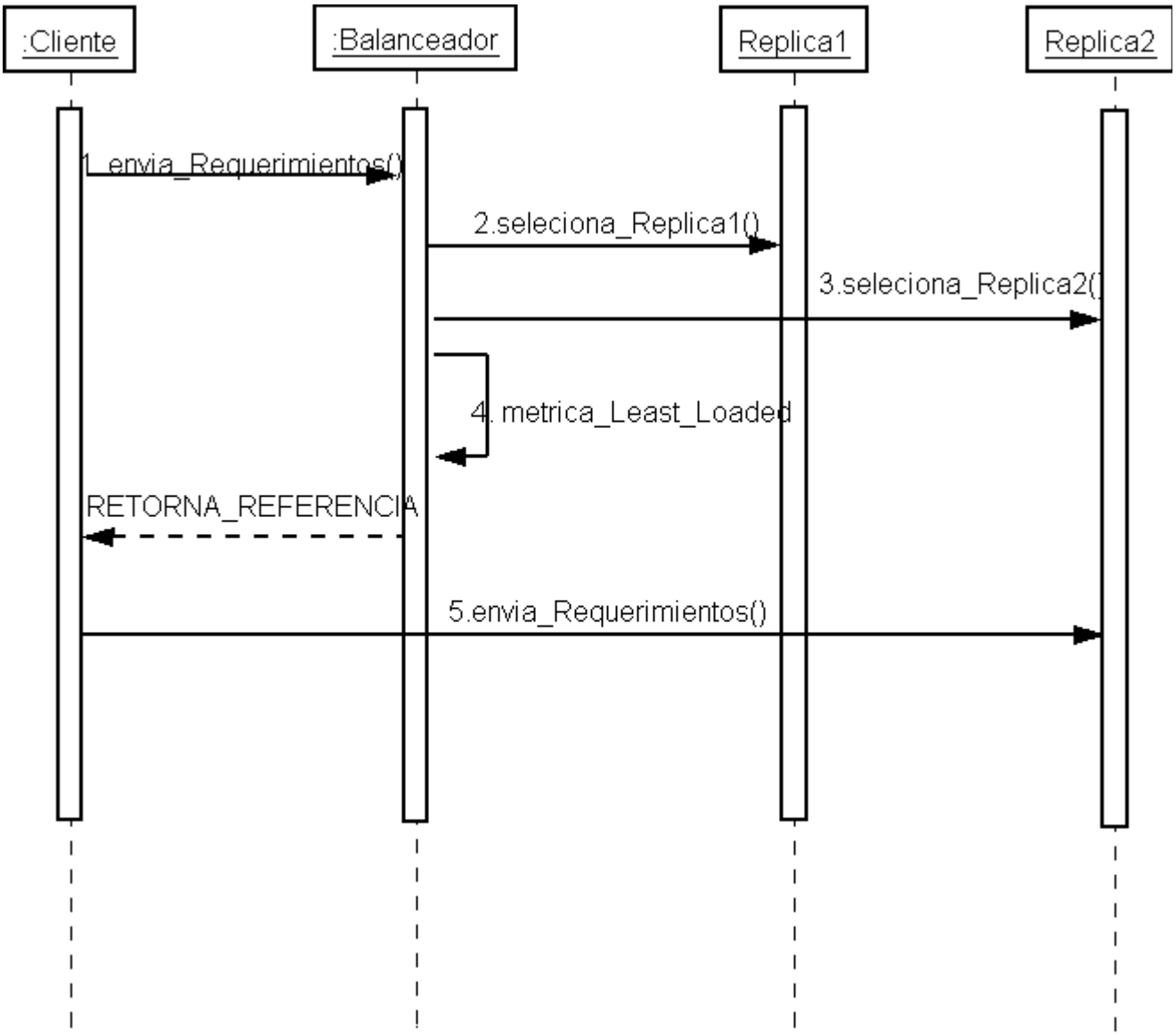


Figura 6.8 El Balanceador de Carga Selecciona una Réplica Aplicando la Estrategia de la Cola Más Corta.

1. El cliente solicita se de respuesta a sus requerimientos.
2. Los requerimientos son interceptados transparentemente por el balanceador de carga. El balanceador de carga selecciona todas las réplicas que se encuentran en un grupo de réplicas.
4. El balanceador de carga elige la métrica *LeastLoaded* cuyo objetivo es seleccionar la réplica que tenga la *Cola Más Corta.* Una vez que el balanceador de carga detecto la réplica con la Cola Más Corta, este envía la referencia de la réplica al cliente.
5. El cliente se comunica con la réplica enviando las peticiones de sus requerimientos.

6.4.4 Estrategia Umbral (Threshold).

Este algoritmo utiliza una política de localización aleatoria para localizar la réplica, una vez que la réplica es seleccionada, se estima la carga de la réplica para determinar si éste es un receptor. Si el balanceador determina que la réplica es un receptor la tarea es transferida a la réplica seleccionada. En otro caso, otro nodo es seleccionado de forma aleatoria y la carga es estimada nuevamente. El número de estimaciones esta limitado por un parámetro llamado *PollLimit*, con el propósito de mantener la sobrecarga (overhead) baja. Si ningún nodo receptor es obtenido en el *PollLimit*, entonces el primer nodo seleccionado es quien deberá ejecutar la tarea. El modelo funcional de esta estrategia es como sigue (ver Figura 6.9):

1. El cliente solicita se de respuesta a sus requerimientos. Los requerimientos son interceptados transparentemente por el balanceador de carga.
2. El balanceador de carga intercepta los requerimientos y elige la métrica umbral para seleccionar las réplicas. El balanceador de carga aplicando la métrica selecciona una réplica aleatoriamente.
3. El balanceador de carga checa si la réplica esta en posibilidades de procesar requerimientos. Si la réplica no se encuentra en posibilidades de procesar requerimientos, se elige una nueva réplica y solicita a la réplica sobrecargada que ya no acepte requerimientos.
4. El balanceador de carga aplicando la métrica umbral selecciona nuevamente una réplica de manera aleatoria.
5. Una vez más el balanceador de carga checa si la réplica esta en posibilidades de procesar requerimientos. Si la réplica se encuentra en posibilidades de procesar requerimientos, se elige como réplica seleccionada y retorna la referencia al cliente. El cliente solicita a la réplica que le de servicio a sus requerimientos.

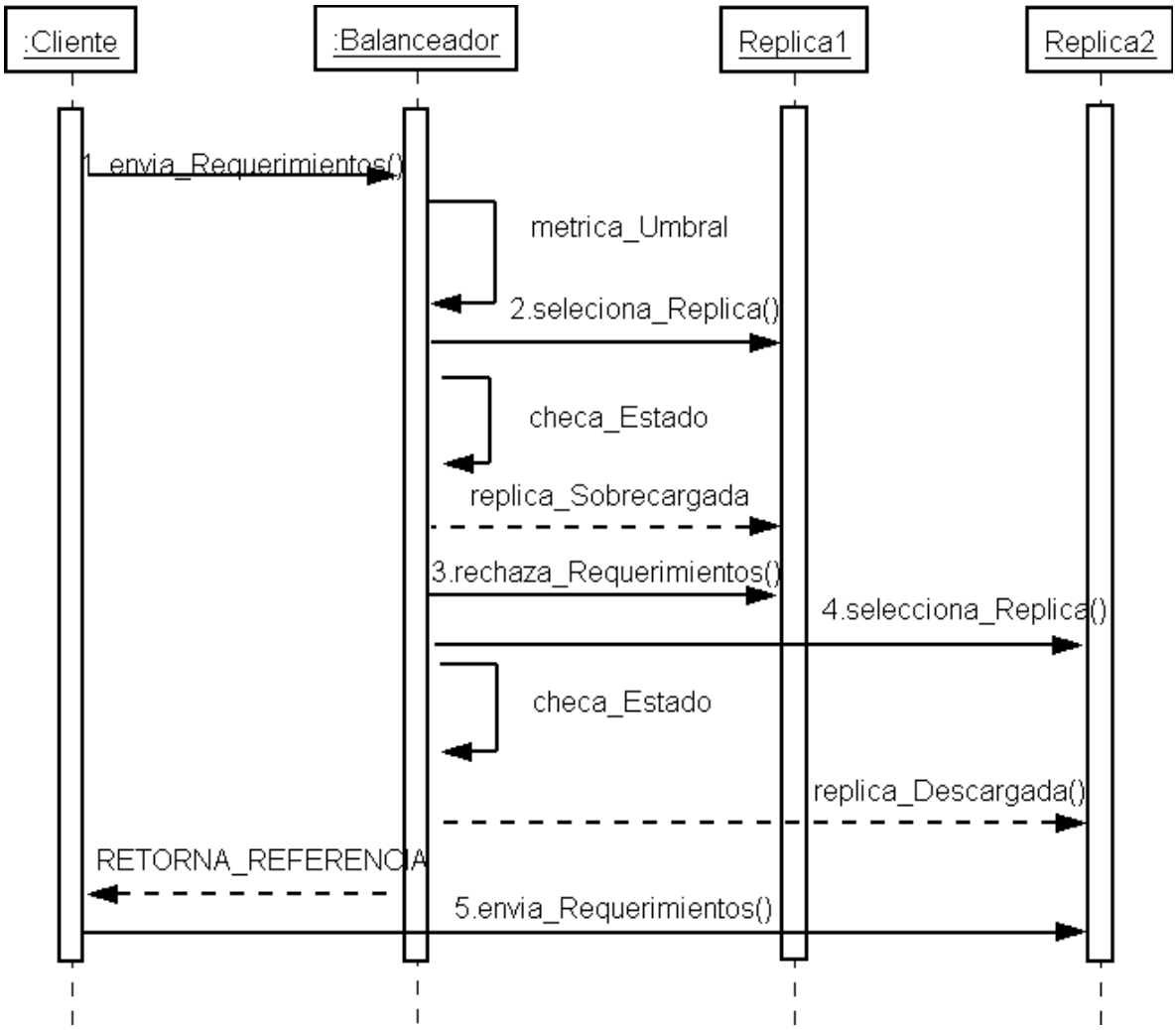


Figura 6.9 El Balanceador de Carga Selecciona una Réplica Aplicando la Estrategia Umbral.

6.4.5 Estrategia Genética.

Una vez que hemos diseñado nuestra estrategia genética (ver sección 6.3), el siguiente paso es conocer como se integra esta estrategia a nuestra arquitectura de balanceo dinámico de carga. Inicialmente la arquitectura de balanceo dinámico utiliza la estrategia de la *Cola Más Corta*. Como habíamos mencionado anteriormente esta estrategia realiza el balanceo dinámico de carga de manera adaptativa, eso significa que esta estrategia utiliza información en tiempo de ejecución para seleccionar la mejor réplica. En un tiempo t la arquitectura de balanceo de carga dinámica checa el estado del sistema y a partir de ese momento la estrategia genética es lanzada planeando la carga que se encuentra en ejecución más la carga que se encuentra en espera. La Figura 6.10 muestra el modelo

funcional de esta estrategia:

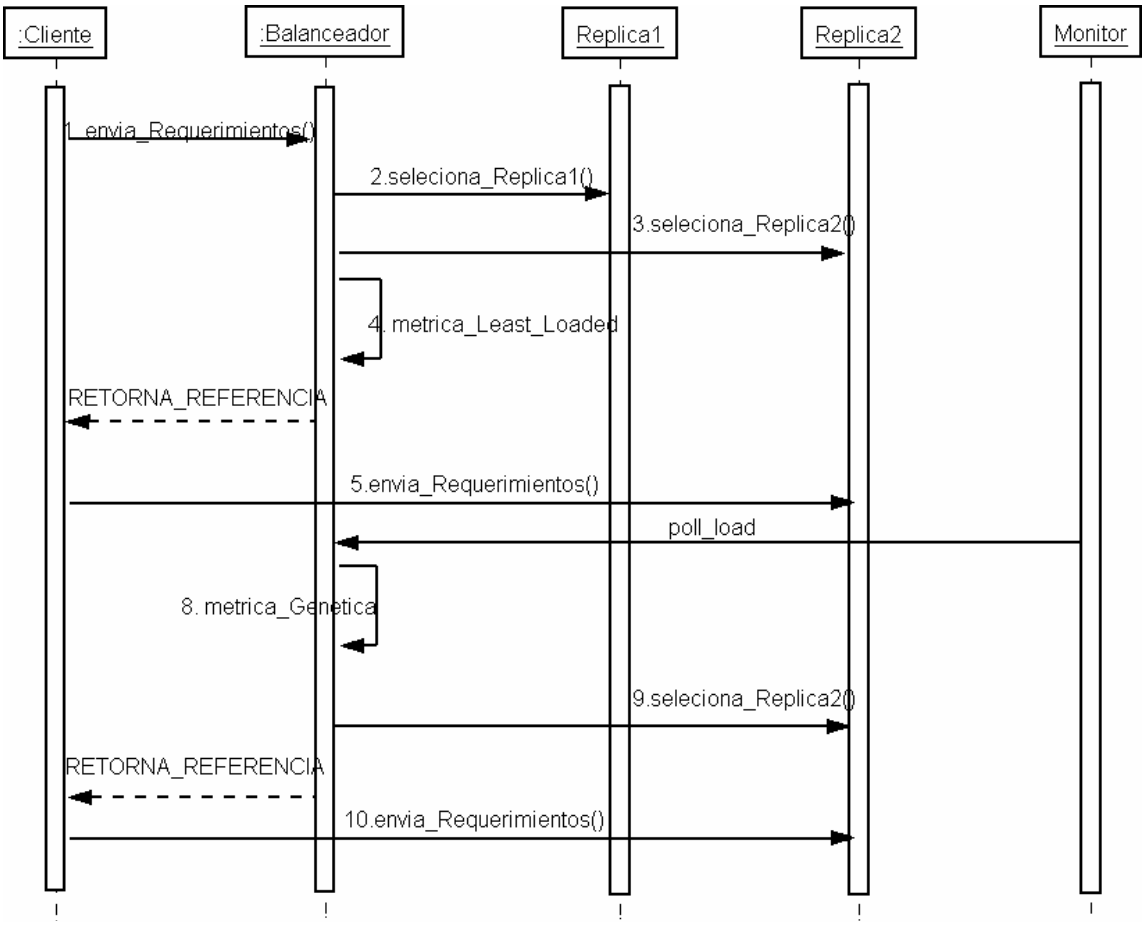


Figura 6.10 El Balanceador de Carga Selecciona una Réplica Aplicando la Estrategia Least_Loaded en Combinación con una Estrategia Genética.

1. El cliente solicita se de respuesta a sus requerimientos.

2. Los requerimientos son interceptados transparentemente por el balanceador de carga. El balanceador de carga selecciona todas las réplicas que se encuentran en un grupo de réplicas.

4. El balanceador de carga elige la métrica *LeastLoaded* cuyo objetivo es seleccionar la réplica que tenga la *Cola Más Corta.* Una vez que el balanceador de carga detectó la réplica con la Cola Más Corta, éste envía la referencia de la réplica al cliente.

5. El cliente se comunica con la réplica enviando las peticiones de sus requerimientos.

8. En un tiempo t el analizador solicita al monitor la carga de las réplicas, en ese momento el analizador de carga decide cambiar de estrategia y selecciona la métrica genética para seleccionar las réplicas, la

métrica permanece hasta que el cliente termine de ejecutar sus requerimientos.
9. Los requerimientos son interceptados transparentemente por el balanceador de carga. El balanceador de carga selecciona todas las réplicas aplicando la métrica genética.
10. El cliente se comunica con la réplica enviando las peticiones de sus requerimientos.

6.5 Un Simple Ejemplo.

Un ejemplo es mostrado a continuación para ilustrar como trabaja el mecanismo de balanceo de carga propuesto. Todos los pasos, comienzan desde el momento que el mecanismo de balanceo de carga es inicializado hasta la asignación de requerimientos.

Paso 1. Asignación de Carga Sin Estrategia Genética. Antes de que el algoritmo genético entre en acción el analizador selecciona las réplicas usando la estrategia LL.

Paso 2. Estado del Sistema Actual. En un tiempo "*t*", el estado del sistema actual es checado. Por ejemplo, nosotros consideramos el siguiente estado.

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

Cada demanda en la réplica representa un grupo de requerimientos preguntando por servicios.

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.

Paso 4. Un Algoritmo Genético es Aplicado. Basado en el conjunto de tareas del paso 2 y 3, un AG es

aplicado de la siguiente manera:

1) Calcula el número de bits por réplica (según apartado 6.3.1).
2) Calcula la composición del cromosoma (según apartado 6.3.2).
3) Genera la población binaria de “N” cromosomas (según apartado 6.3.3).
4) Evaluar cada cromosoma de acuerdo a su función objetivo (ver apartado 6.3.4.2).
5) Generar el proceso evolutivo aplicando elitismo, cruzamiento en un punto y mutación (ver apartado 6.3.4.1).

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 6.6 Información de Carga en un Tiempo t + 1.

Paso 5. Asignación de Tareas. Después de “N” generaciones el AG determino que la mejor planeación es la que muestra la Tabla 6.6.

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

6.6 Análisis Comparativo de las Estrategias.

Enseguida se presenta un comparativo de las cinco estrategias descritas previamente, se tomaron los índices de desempeño utilizados por Andrzej Goscinski en [Goscinski 94]:

Tiempo de Respuesta Total

Rendimiento del Sistema (throughput).

Los índices son tomados como métricas de comparación, sin embargo no se descarta la posibilidad de considerar otros factores.

6.6.1 Balanceo Dinámico de Carga Homogéneo.

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

Plataforma de Hardware.

Actualmente la arquitectura tiene 9 PC´s para los clientes, 4 PC´s que funcionan como réplicas, 1 PC para el analizador de carga y 1 PC más que funciona como monitor de carga. Todas las PC´s tienen sistema operativo Windows XP, 128 Mbytes en RAM y son Pentium IV en 2.6 GHz, conectadas a una Red LAN base 100 usando un Switch CISCO de 24 Puertos.

El balanceo dinámico de caga homogéneo se llevo a cabo de la siguiente manera:

Primero. Se aplicó utilizando un mismo tipo de requerimiento, arquitectura y velocidad de procesamiento.

Tiempo Total (Min) G = 10 bytes					
No. de archivos	Genética	Aleatoria	Round Robin	CMC	Umbral
4500	0.844	1.006	0.904	0.907	1.049
9000	1.813	2.267	1.875	2.134	2.663
18000	4.290	4.380	3.888	3.732	9.479
27000	5.133	6.788	5.393	5.584	10.177
36000	7.559	8.899	7.514	6.931	23.574
45000	8.402	10.890	9.335	8.818	37.170

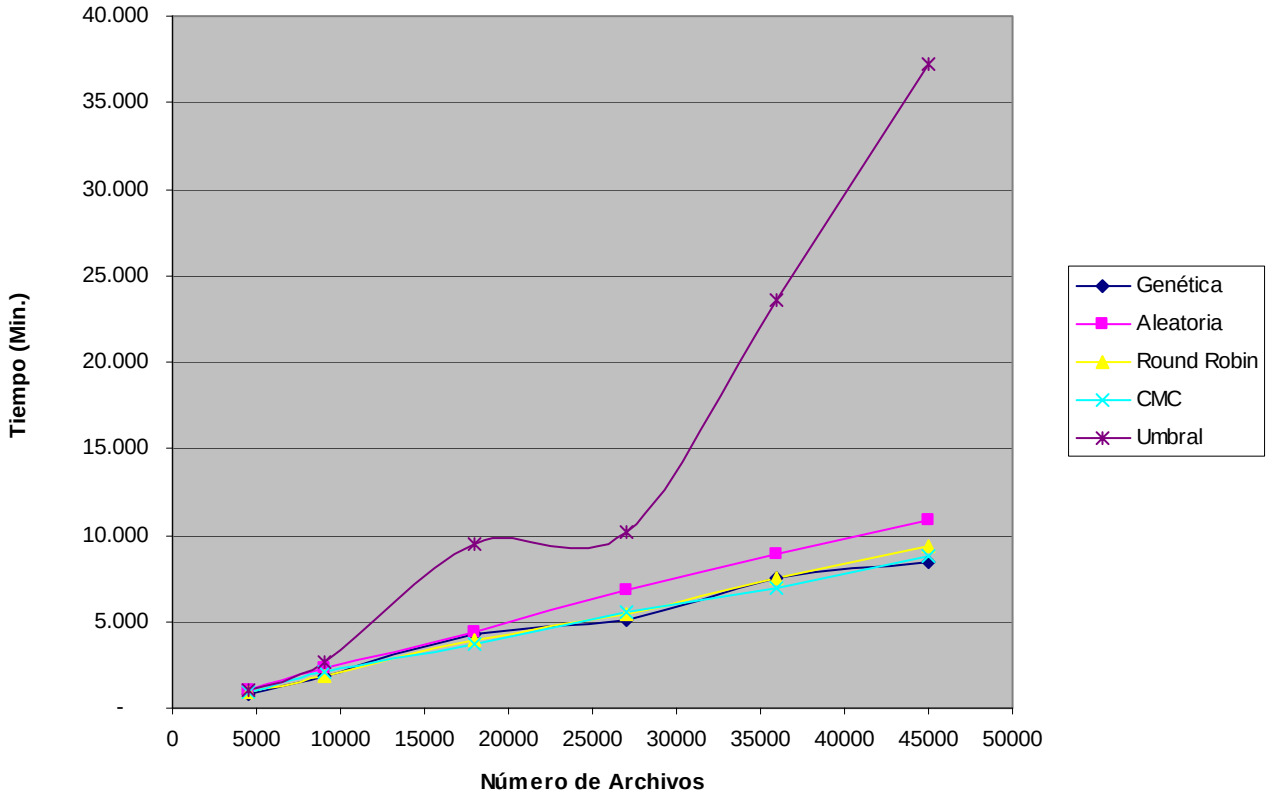


Figura 6.11 Tiempo Total (Balanceo Dinámico de Carga Homogéneo con Granularidad = 10 bytes).

Segundo. Se implementó una aplicación cuyo objetivo es la transferencia de archivos

Tercero. Se evaluaron 4500, 9000, 18000, 27000, 36000 y 45000 archivos con granularidad constante.

Rendimiento del Sistema (Mb/Min) G = 10 bytes					
No. de archivos	Genética	Aleatoria	Round Robin	CMC	Umbral
4500	0.051	0.043	0.047	0.046	0.041
9000	0.045	0.038	0.046	0.039	0.032
18000	0.038	0.039	0.044	0.045	0.017
27000	0.048	0.037	0.047	0.045	0.024
36000	0.043	0.038	0.044	0.049	0.013
45000	0.050	0.039	0.045	0.048	0.011

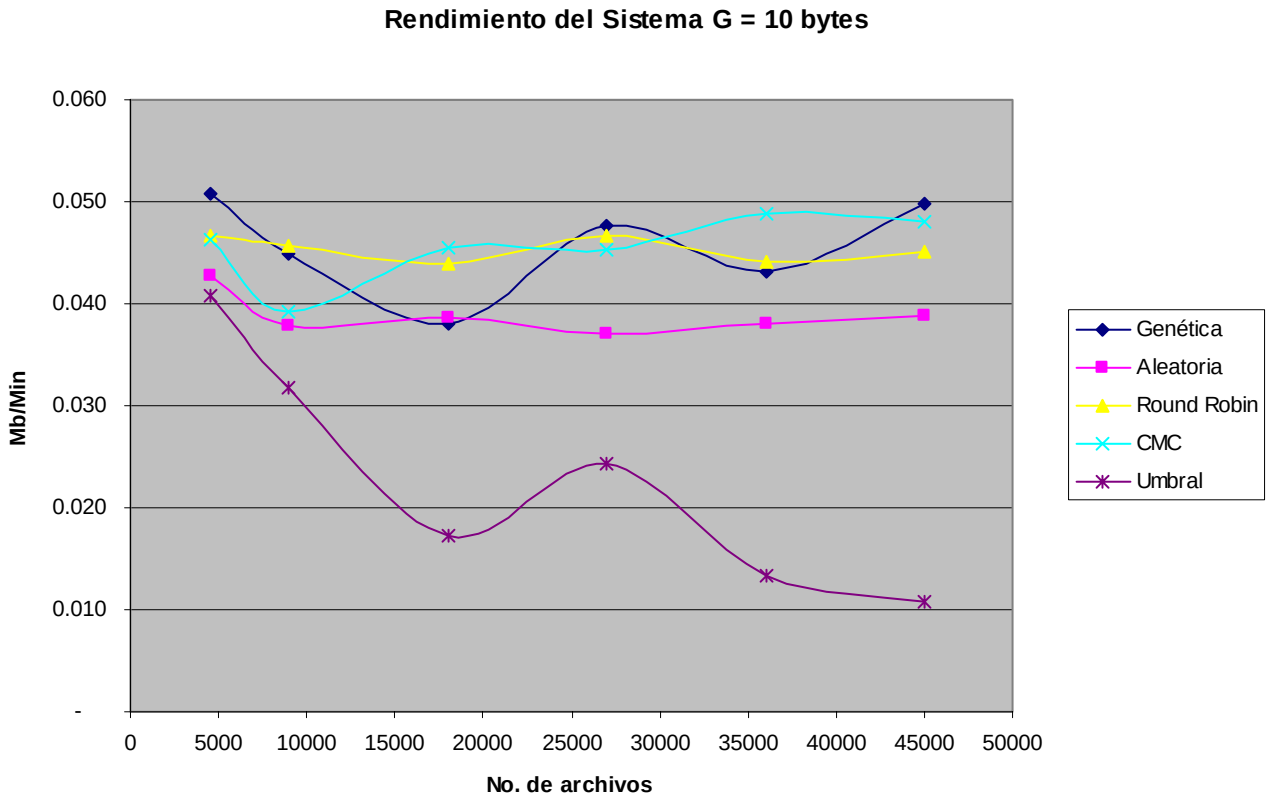


Figura 6.12 Rendimiento del Sistema (Balanceo Dinámico Homogéneo con Granularidad = 10 bytes).

Como podemos observar en la Figura 6.11, las estrategias Aleatoria, Round Robin, Cola Más Corta (CMC) y Genética mantienen respuestas muy similares cuando la carga es ligera y la granularidad de los archivos es pequeña. También observamos como la estrategia Umbral no es una buena opción cuando manejamos este tipo de carga, ya que genera tiempos de respuesta demasiado prolongados. A medida que se incrementa la carga, se observa como las estrategias Round Robin y Aleatoria dejan de ser una buena elección, convirtiendo las estrategias Genética y CMC como la mejor opción. En la Figura 6.12 observamos como las estrategias Round Robin y CMC pueden ser una buena opción cuando queremos

aprovechar la utilización de recursos, sin embargo no podemos decir lo mismo de las otras estrategias ya que su comportamiento se presenta muy inestable en este tipo de aplicaciones.

Tiempo Total (Min) G = 100 bytes					
No. de archivos	Genetica	Aleatoria	Round Robin	CMC	Umbral
4500	3.348	5.094	2.944	2.784	3.310
9000	5.676	11.347	6.050	5.676	6.613
18000	10.719	21.942	12.268	12.619	16.679
27000	18.688	32.093	18.378	18.463	27.837
36000	22.753	42.341	24.316	24.410	31.099
45000	30.007	53.473	30.480	30.484	62.240

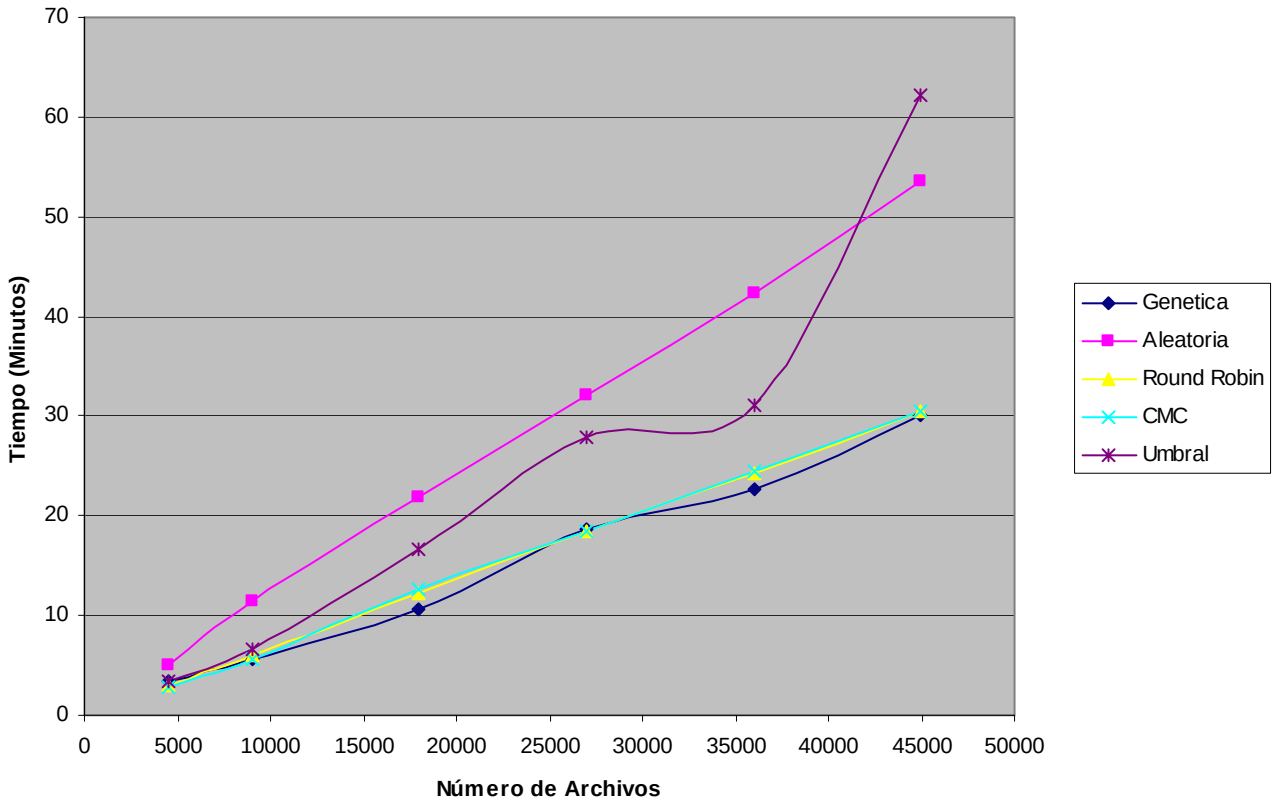


Figura 6.13 Tiempo Total (Balanceo Dinámico de Carga Homogéneo con Granularidad = 100 bytes).

Cuarto. Se tomó como base el número de archivos del punto anterior y se elevó el peso del archivo a 100kb, una vez elevado el peso del archivo la granularidad se mantuvo constante.

La Figura 6.13 muestra claramente como las estrategias Round Robin, CMC y Genética se separan de las otras dos estrategias generando un mejor tiempo de respuesta. También observamos como la estrategia Umbral comienza a ganar terreno en comparación con la estrategia Aleatoria, la razón es muy sencilla, el algoritmo Aleatorio debido a su naturaleza no-adaptativa no checa el estado en tiempo de ejecución, esto por consecuencia genera tiempos de respuesta demasiado prologados cuando los archivos comienzan a tener un peso considerable.

Quinto. Se consideró nuevamente el número de archivos de los dos puntos anteriores y se elevó el peso del archivo a 512kb, después de elevar el peso del archivo la granularidad se mantuvo constante.

La grafica de la Figura 6.14 ejemplifica el tiempo de respuesta global de las estrategias cuando consideramos la granularidad igual a 512 Kbytes. Como podemos observar en dicha figura, las estrategias Aleatoria y Round Robin son las que peores tiempos de desempeño generan y es lógico ya que por su naturaleza estas estrategias no verifican el estado del sistema en tiempo de ejecución.

La estrategia Umbral se puede visualizar en la parte intermedia de la gráfica, a pesar de que esta estrategia verifica el tiempo de respuesta en tiempo de ejecución, la estrategia tiende a degenerarse conforme la carga aumenta. Las estrategias CMC y Genética generan los mejores tiempos de respuesta globales, de hecho la estrategia Genética es el que mejor tiempo genera la razón es simple la estrategia genética se hace cada vez más rentable a medida que la granularidad del archivo aumenta.

Tiempo Total (Min) G = 512 Kb					
No. de archivos	Genética	Aleatoria	Round Robin	CMC	Umbral
4500	5.053	17.016	13.014	5.025	12.017
9000	8.002	24.043	24.043	8.682	33.690
18000	16.065	66.006	52.041	18.029	40.006
27000	24.075	102.002	71.011	26.045	68.052
36000	30.008	136.023	93.010	32.020	84.009
45000	38.025	181.688	123.681	39.690	89.345

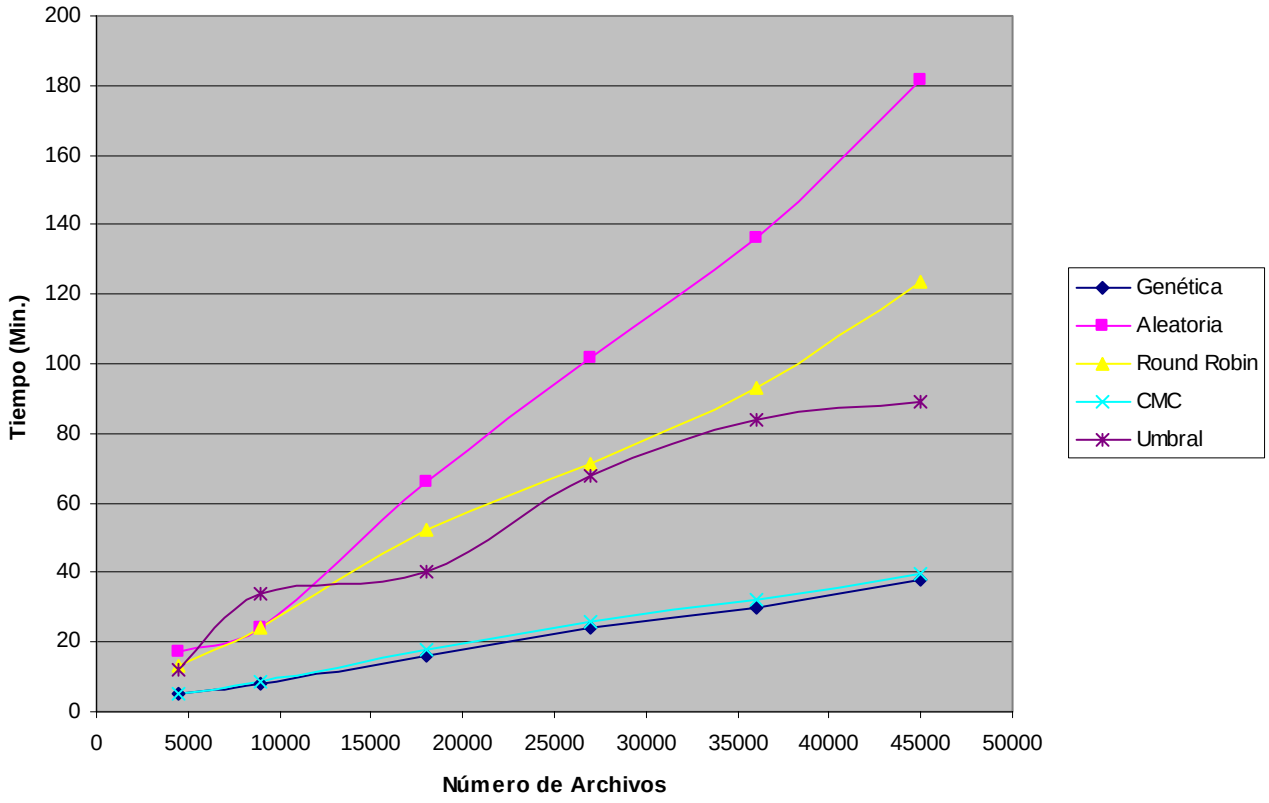


Figura 6.14 Tiempo Total (Balanceo Dinámico Homogéneo con Granularidad = 512 kbytes).

La utilización de recursos se puede ver reflejada en la Figura 6.15. Como podemos observar en la figura, la estrategia Genética y CMC son las que más aprovechamiento de recursos tienen, esto es obvio ya que estas estrategias son las que mejores tiempos de respuesta generan.

Como pudo observarse, el comparativo se evaluó utilizando diferentes métricas de carga y se ilustró como bajo ciertas condiciones de carga, algunas estrategias generaron mejores resultados que otras. Se

determinó que la estrategia más adecuada para elegir un algoritmo adaptativo o no adaptativo depende del tipo granularidad del archivo y de la carga que se genere.

Rendimiento del Sistema (Mb/Min) G = 512 Kb					
No. de archivos	Genética	Aleatoria	Round Robin	CMC	Umbral
4500	444.540	134.990	175.132	442.223	194.234
9000	562.341	133.986	178.572	518.604	178.572
18000	541.215	136.352	172.941	499.208	224.968
27000	524.072	132.350	190.110	506.280	198.378
36000	595.343	132.331	193.528	544.750	214.264
45000	573.903	125.156	181.954	566.973	252.325

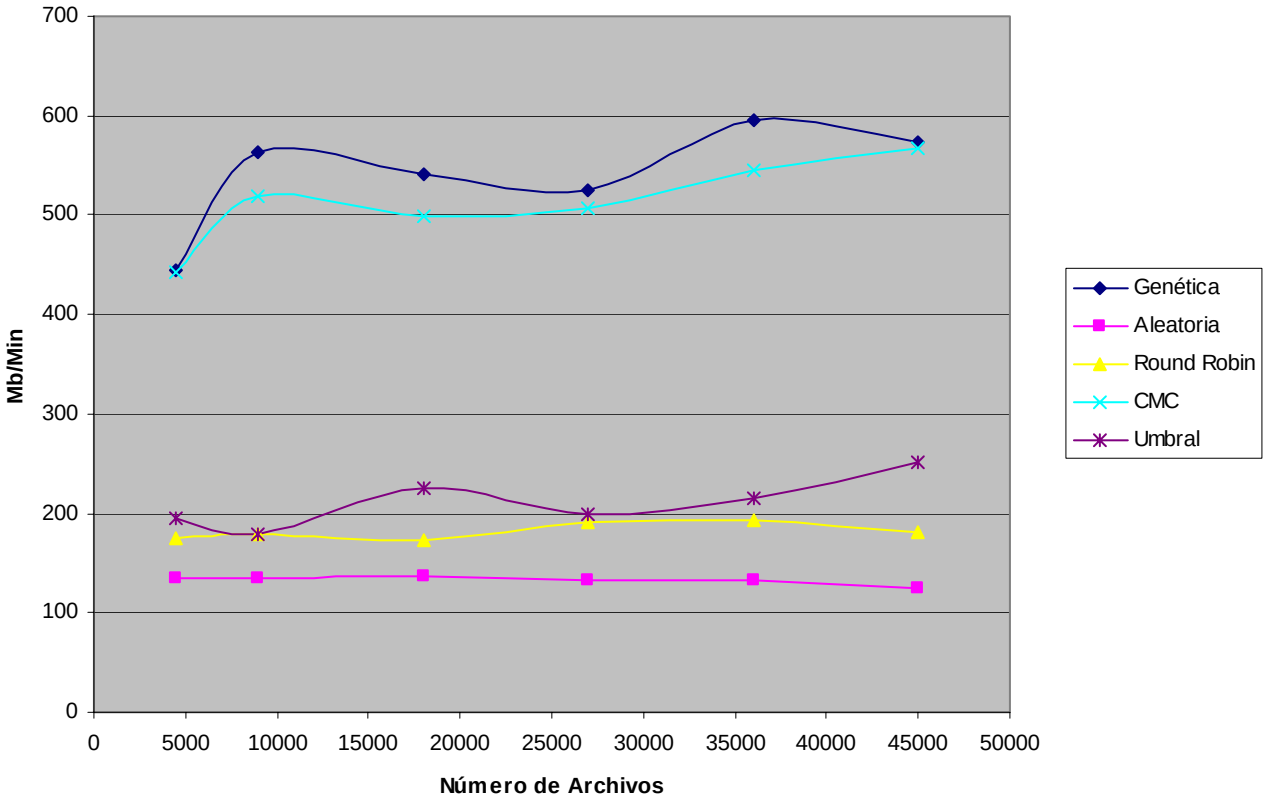


Figura 6.15 Rendimiento del Sistema (Balanceo Dinámico Homogéneo con Granularidad = 512kbytes).

6.6.2 Balanceo Dinámico de Carga Heterogéneo.

Plataforma de Hardware.

Actualmente la arquitectura tiene 9 PC´s para los clientes con sistema operativo Windows XP, 128 Mbytes en RAM y son Pentium IV en 2.6 Ghz, 4 réplicas, 1 réplica mantiene las mismas características que los clientes, 2 réplicas tienen sistema operativo Linux Mandrake versión 9.2 una tiene 256 Mbytes en RAM con procesador Pentium III a 850 GHz la otra tiene 128 Mbytes de RAM y es Pentium IV en 2.6 GHz. La última réplica es un SUN NECTRA X1 con sistema operativo Solares 8.0. El analizador de carga y el monitor son PC´s que mantienen las mismas características que los clientes. Todas las PC´s son conectadas a una Red LAN base 100 usando un Switch CISCO de 24 Puertos.

El balanceo se llevó a cabo de la siguiente manera:

Primero. El balanceo de carga heterogéneo fue aplicado usando diferentes sistemas operativos, arquitecturas y velocidades de procesamiento. Además de aplicar diferentes pesos en los requerimientos.

Segundo. Se implementó una aplicación cuyo objetivo es la transferencia de archivos.

Tercero. Se evaluaron 4500, 9000, 18000, 27000, 36000 y 45000 archivos con granularidad constante.

Cuarto. Se tomó como base el peso del archivo a 100kb, una vez considerado el peso del archivo la granularidad se mantuvo constante.

Tiempo Total (Min) G = 100 Kb					
No. de archivos	Genética	Aleatoria	Round Robin	CMC	Umbral
4500	3.386	5.568	3.881	3.035	4.262
9000	6.296	14.919	6.768	6.296	7.568
18000	12.672	27.548	14.816	13.242	16.543
27000	19.409	43.394	22.965	19.409	25.449
36000	25.454	51.793	29.633	24.491	36.725
45000	32.213	69.596	36.412	31.799	61.935

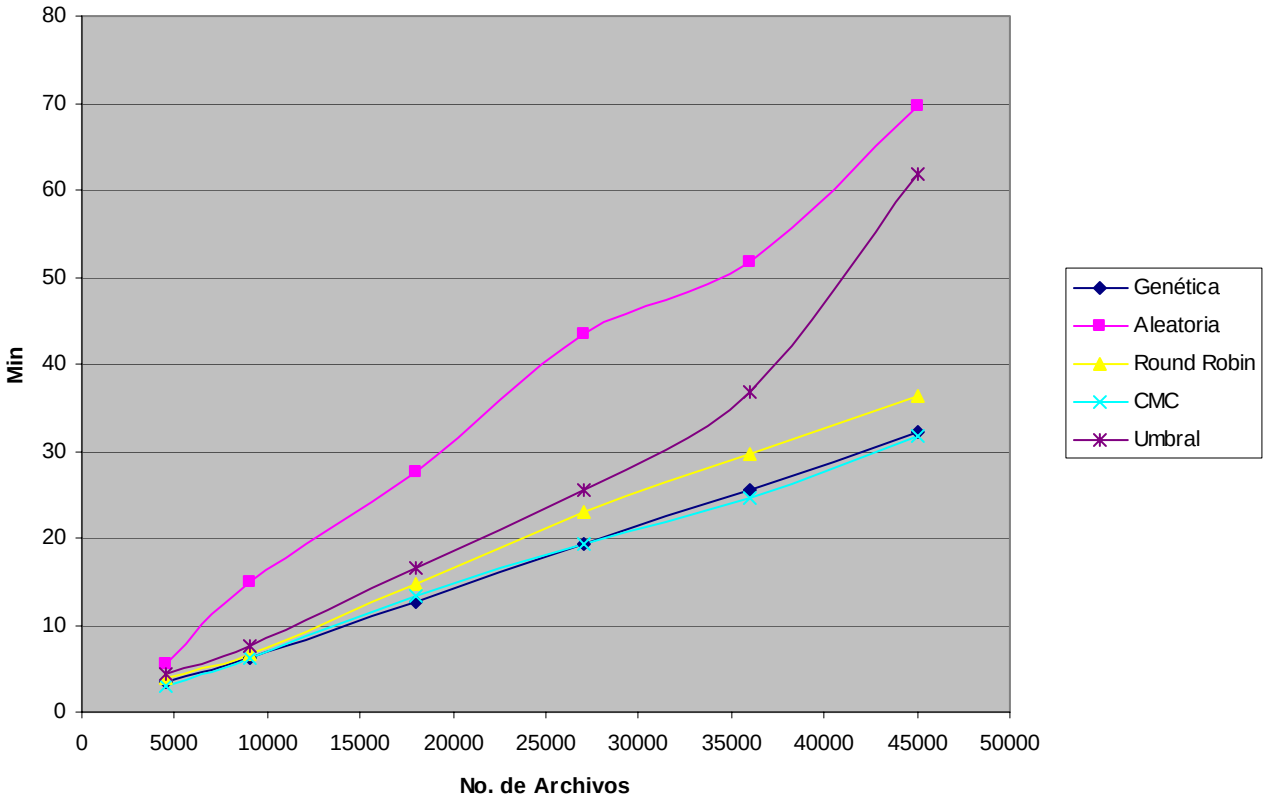
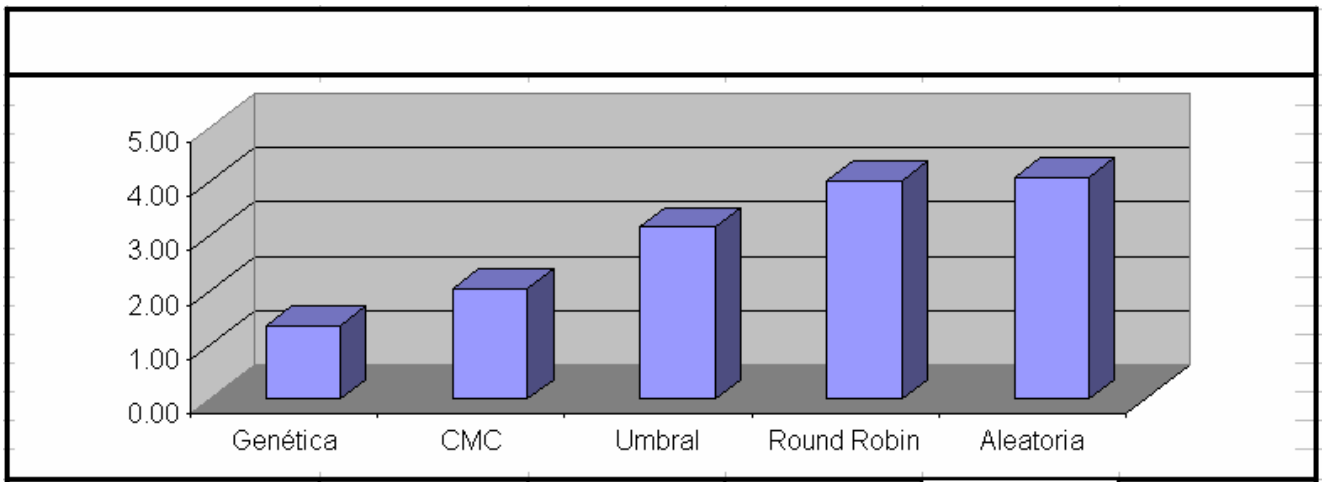


Figura 6.16 Tiempo Total (Balanceo Dinámico Heterogéneo con Granularidad = 100kbytes).

La Figura 6.16 muestra el comportamiento de las estrategias en un ambiente heterogéneo, podemos observar claramente como de las dos estrategias no adaptativas, la que peores resultados genera es la estrategia Aleatoria. Por otra parte, de las tres estrategias que checan el estado en tiempo de ejecución (adaptativas) la CMC y la estrategia Genética generan tiempos de respuesta muy similares, de hecho observamos que a medida que aumenta la carga estas dos estrategias se separan claramente generando tiempos de respuesta más satisfactorios en comparación con las otras estrategias.

Quinto. 900 archivos con granularidad gruesa fueron evaluados. La Figura 6.17 muestra el tiempo total de las diferentes estrategias en un grupo de réplicas heterogéneas cuando el peso del archivo es de 5Mbytes.

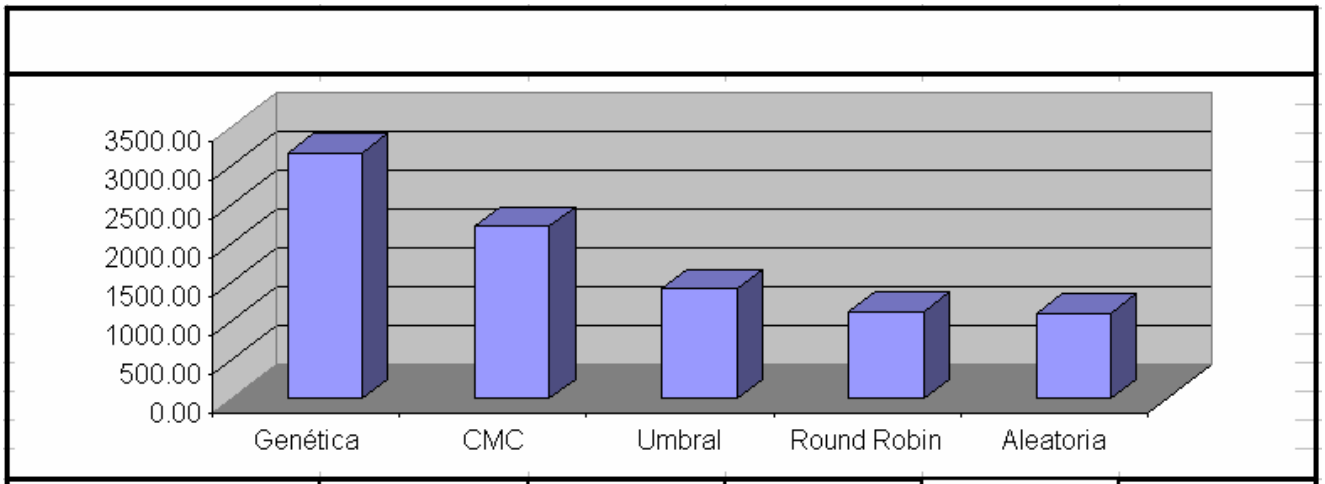


Estrategia	Genética	CMC	Umbral	Round Robin	Aleatoria
Tiempo(Min.)	1.35	2.03	3.17	4.02	4.08

Figura 6.17 Tiempo Total (Balanceo Dinámico de Carga Heterogéneo con Granularidad = 5Mbytes)

Observamos como la estrategia genética genera considerables ventajas comparada con las otras estrategias, la estrategia genética mejora el tiempo de respuesta global en casi medio minuto comparado con la estrategia que usa la CMC y en más de dos minutos si la comparamos con la estrategia Umbral, estas estrategias al igual que la estrategia Genética checan el estado de la carga en tiempo de ejecución (estrategias adaptativas). También observamos que la estrategia genética es tres veces más rápida comparada con la estrategia que usa estrategias no adaptativas (Aleatorias y Round Robin).

El uso de recursos puede verse reflejado en la Figura 6.18, la estrategia genética es la que mejor aprovecha los recursos y esto es correcto, ya que es la que mejores tiempos de respuesta genera. Tomando en cuenta los resultados previamente descritos podemos seleccionar la estrategia genética como la mejor cuando la carga es elevada y se utiliza una granularidad gruesa.



Estrategia	Genética	CMC	Umbral	Round Robin	Aleatoria
Throughput(Mb/Min)	3156.36	2216.90	1418.68	1120.61	1102.99

Figura 6.18 Rendimiento del Sistema (Balanceo Dinámico de Carga Heterogéneo con Granularidad = 5Mbytes).

6.7 Conclusiones.

Un sistema distribuido emplea el servicio de balanceo de carga para mejorar el rendimiento asegurándose que las cargas entre las réplicas sean lo más uniforme posible. CORBA provee soluciones para muchos cambios en sistemas distribuidos, tales como predictibilidad, seguridad, transacciones y tolerancia a fallas, pero CORBA aún carece de soluciones o mecanismos estándar para intentar balancear la carga.

El balanceo de carga basado en CORBA es importante por que permite mejorar el tiempo de respuesta global y el aprovechamiento óptimo de recursos de las aplicaciones distribuidas. Se analizó el desempeño de una arquitectura capaz de ejecutar balanceo dinámico de carga adaptativo y no adaptativo bajo CORBA utilizando JAVA-IDL.

La arquitectura usa estrategias n*o adaptativas* para el atado de los requerimientos y esta estrategia es aplicada por toda la vida de los clientes. Este tipo de estrategias tienen un buen desempeño en cargas moderadas pero un mal desempeño en aplicaciones con cargas pesadas. La arquitectura elige estrategias Round-Robin o Aleatorias (Random), para seleccionar una réplica en donde se procesará el requerimiento.

La arquitectura puede usar estrategias *adaptativas* que utilicen información en tiempo de ejecución (carga de trabajo en las réplicas, requerimientos en espera, etc.) para seleccionar la mejor réplica, que procese los requerimientos de los clientes. Este tipo de estrategias tienen un buen desempeño en aplicaciones que tienen cargas sumamente pesadas. En este tipo de estrategia la arquitectura elige los algoritmos Umbral, Cola Más Corta y Genética.

Capítulo 7. Análisis Comparativo de Estrategias en el Procesamiento Distribuido de Imágenes.

En este capítulo se realiza un análisis comparativo de las estrategias de balanceo de carga en el procesamiento distribuido de imágenes. Evaluamos las estrategias aplicando diversos operadores individuales (negativo, escala de grises, la potencia para abrillantar u oscurecer una imagen y la binarización de la imagen) [Lyon 99, González 95 y Pajares 04] para transformar el valor de cada píxel en la imagen de entrada y generar una nueva imagen de salida con los píxeles transformados. Evaluamos las estrategias implementado el algoritmo de Zhang y Suen [Lyon 99] con el propósito de adelgazar (esqueletizar) los contornos de una imagen binaria. Por último implementamos operaciones basadas en múltiples puntos (vecindad de píxeles) con el propósito de realizar la detección de bordes basada en la convolución [Lyon 99 y González 95]. Evaluamos las estrategias implementado el operador Prewitt, Laplaceano y Roberts con el propósito de detectar bordes para delinear los límites de los objetos en la imagen.

7.1 Introducción.

El campo del tratamiento digital de imágenes está en continua evolución, González y Woods dicen que durante los últimos 5 años ha aumentado significativamente el interés en la morfología de las imágenes, el procesamiento de imágenes en color, la compresión, el reconocimiento de imágenes y los sistemas inteligentes de análisis de imágenes [González 95]. Las técnicas de procesamiento digital de imágenes se emplean actualmente para resolver problemas muy diversos. En medicina por ejemplo, los procesamientos realzan el contraste o codifican los niveles de intensidad de colores para facilitar la interpretación de las imágenes de rayos X y de otras imágenes biomédicas. Los geógrafos emplean las mismas o similares técnicas para estudiar los patrones de contaminación a partir de imágenes aéreas o de satélites. En arqueología, los métodos de procesamiento de imágenes han servido para restaurar con éxito imágenes borrosas que eran los únicos registros existentes de piezas extrañas perdidas o dañadas después de haber sido fotografiadas. En la física y en campos afines, las técnicas de procesamiento realzan de forma rutinaria imágenes de experimentos en áreas como los plasmas de alta energía y la microscopía del electrón. De forma similar, las técnicas de procesamiento de imágenes se aplican con éxito en astronomía, biología, medicina nuclear, investigaciones judiciales, defensa y aplicaciones industriales.

7.2 Operaciones Individuales.

El proceso consiste en obtener el valor del píxel de una localización dada en la imagen, modificándolo por una operación lineal o no lineal y colocando el valor del nuevo píxel en la correspondiente localización de la nueva imagen. El proceso se repite para todas y cada una de las localizaciones de los píxeles en la imagen original, Figura 7.1

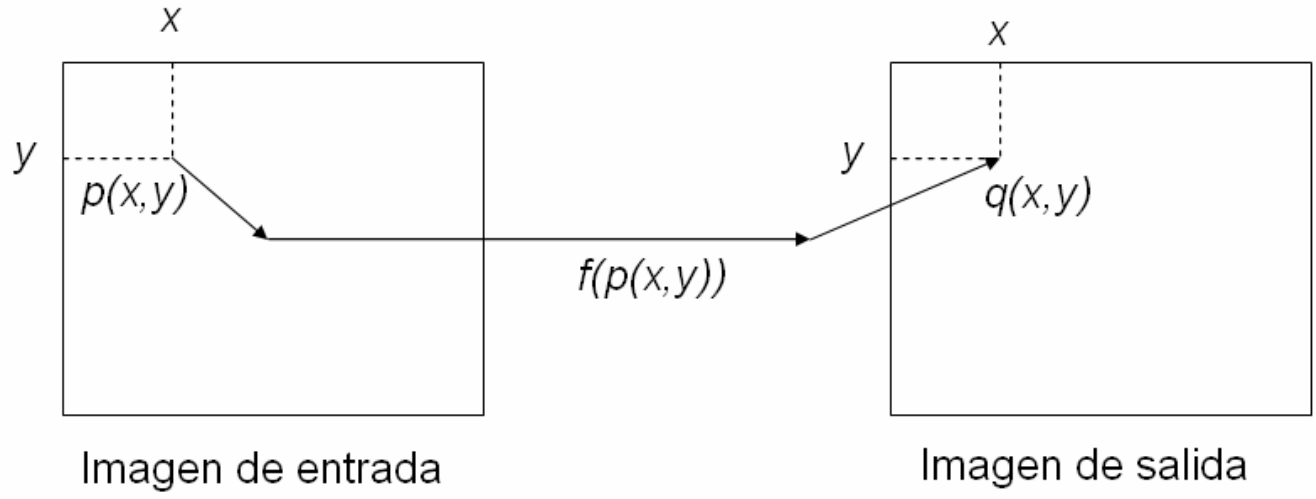


Figura 7.1 Operación Individual.

Como se aprecia en la Figura 7.1, el operador individual es una transformación uno a uno. El operador f se aplica a cada píxel en la imagen o sección de la imagen y la salida depende únicamente de la magnitud del correspondiente píxel de entrada; la salida es independiente de los píxeles adyacentes. La función transforma el valor de cada píxel en la imagen y el nuevo valor se obtiene a través de la ecuación:

$$(v_{ij})' = f(v_{ij}) \tag{7.1}$$

La función f puede ser un operador lineal o no lineal. El proceso matemático es relativamente simple. La imagen resultante es de la misma dimensión que la original.

7.2.1 Operador Inverso o Negativo.

Este operador crea una imagen de salida que es la inversa de la imagen de entrada (ver Figura 7.2). Este operador es útil en diversas aplicaciones tales como imágenes médicas. Para una imagen con valores en el rango de 0 a 255 la función de transformación resulta ser:

$$f(v_{ij}) = 255 - v_{ij} \tag{7.2}$$

Figura 7.2 Imagen Inversa o Negativa Derivada de la Imagen Original [Lyon 99].

El fragmento de código para implementar (7.2) es como sigue:

```
public void negate() {
        for (int x=0; x < width; x++)
                for (int y=0; y < height; y++) {
                r[x][y] = (short) (255 - r[x][y]);
                g[x][y] = (short) (255 - g[x][y]);
                b[x][y] = (short) (255 - b[x][y]);
        }
        short2Image();
}
```

7.2.2 Función de Potencia (Pow) para Abrillantar u Oscurecer una Imagen.

La función de potencia debe ser descrita en términos de un sistema de coordenadas no normalizadas, asumiendo que la función f tiene un rango de 0..255 :

$$f(v_{ij}) = 255\left(\frac{v_{ij}}{255}\right)^{pow} \tag{7.3}$$

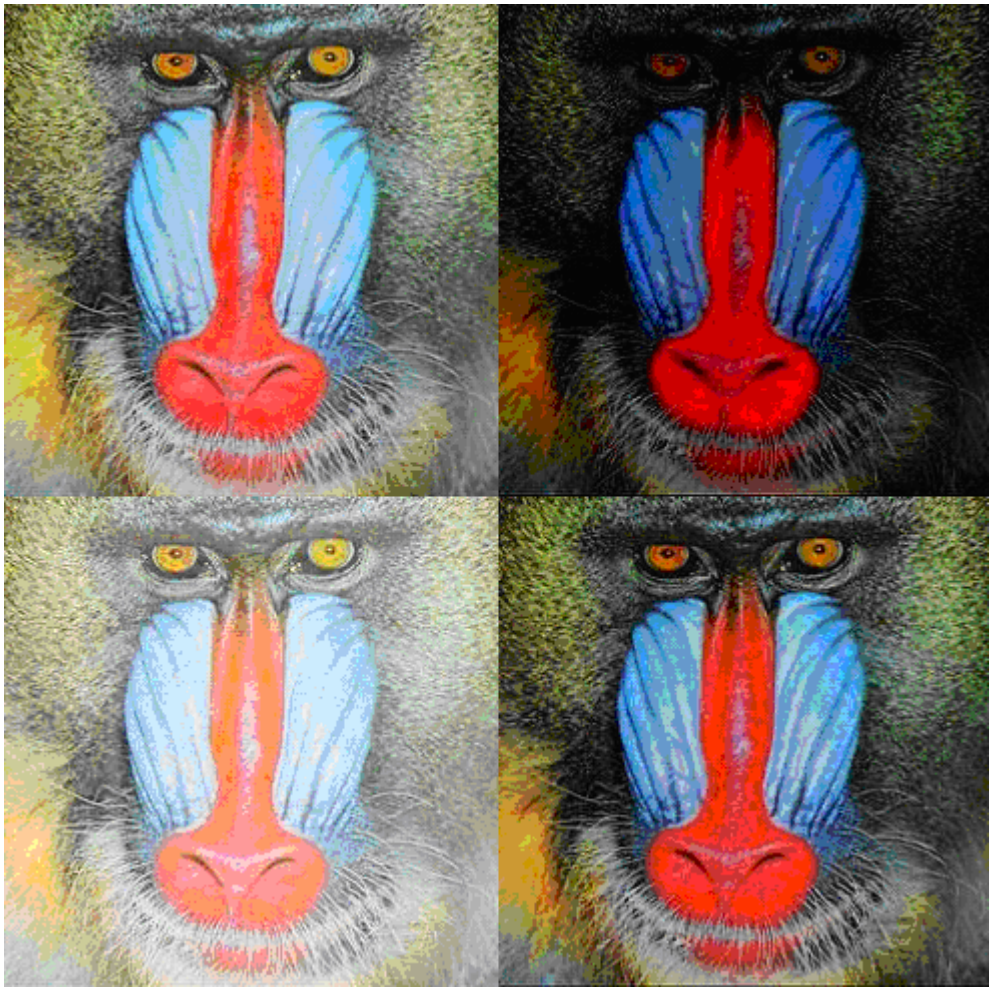


Figura 7.3 Imagen Abrillantada u Oscurecida como Resultado de Aplicar $pow < 1$ o $pow > 1$ *[Lyon 99].*

Si $pow < 1$ la imagen tiende a abrillantarse, los resultados de aplicar (7.3) con $pow := 0.9$ es abrillantar la imagen ligeramente, aplicaciones repetidas hacen que la imagen aparezca brillante (ver

Figura 7.3). En la parte superior derecha de la Figura mostramos los resultados de la imagen cuando $pow > 1$. La parte inferior izquierda de la Figura muestra la imagen obtenida al aplicar $pow < 1$ en repetidas transformaciones, la parte inferior derecha tiene una sola repetición de $pow := 0.9$.

El fragmento de código para implementar (7.3) es como sigue:

```
public void powImage(double p) {
        for (int x=0; x < width; x++)
                for (int y=0; y < height; y++) {
                        r[x][y] = (short)
                         (255 * Math.pow((r[x][y]/255.0),p));
                        g[x][y] = (short)
                        (255 * Math.pow((g[x][y]/255.0),p));
                        b[x][y] = (short)
                        (255 * Math.pow((b[x][y]/255.0),p));
        }
        short2Image();
}
```

7.2.3 Escala de Grises.

Este operador crea una imagen de salida que es el promedio RGB de la imagen de entrada. Para una imagen con valores en el rango de 0 a 255 la función de transformación resulta ser:

$$f(v_{ij}) = \frac{(R_{ij} + G_{ij} + R_{ij})}{3} \tag{7.4}$$

En la parte derecha de la Figura 7.4 se muestra la imagen con escala de grises como resultado de aplicar la formula (7.4).

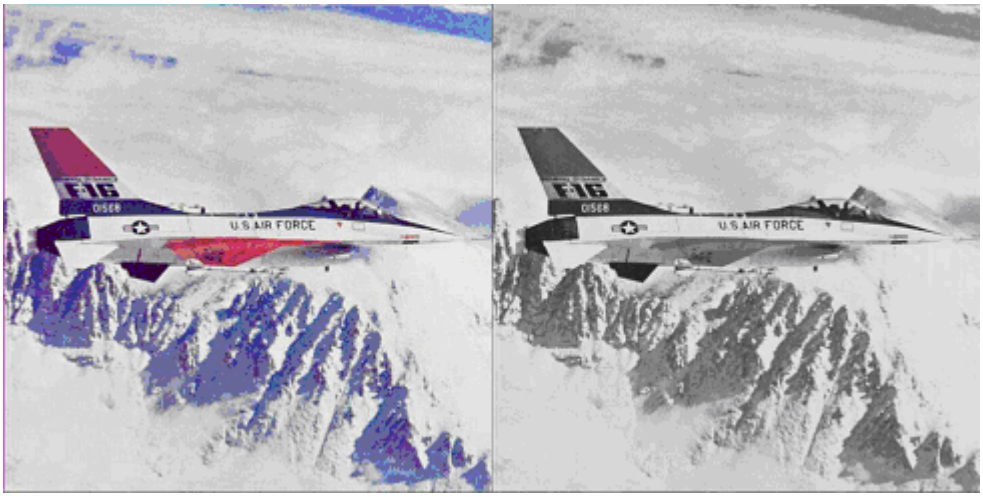


Figura 7.4 Imagen con Escala de Grises Derivada de la Imagen Original [Lyon 99].

El fragmento de código para implementar (7.4) es como sigue:

```
public void gray() {
        for (int x=0; x < width; x++)
                for (int y=0; y < height; y++) {
                        r[x][y] = (short)
                                        ((r[x][y] + g[x][y]  + b[x][y]) / 3);
                                        g[x][y] = r[x][y];
                                        b[x][y] = r[x][y];
        }
        short2Image();
}
```

7.2.4 Binarización.

Esta clase de transformación crea una imagen de salida binaria a partir de una imagen de entrada, donde el nivel de transformación está dado por el parámetro de entrada $p1$ (ver Figura 7.5). La función de transformación es la siguiente.

$$q = \begin{cases} 0 & \text{para} \quad p<=p1 \\ 255 & \text{para} \quad p>p1 \end{cases} \tag{7.5}$$

En la parte derecha de la Figura se muestra la imagen binarizada como resultado de aplicar $p1 = 128$.

Figura 7.5 Imagen Binarizada como Resultado de Aplicar $p1 = 128$ *[Lyon 99].*

El fragmento de código es como sigue:

```
public void binarizar() {
        short umbral=128;
        short negro=0;
        short blanco=255;
        short valor;
                for (int y=0; y < height  ;y++) {
                        for (int x=0;x< width;x++) {
                                r[x][y] =(short)((r[x][y]+g[x][y]+b[x][y])/3);
                               valor= r[x][y];
                                if (valor < umbral)
                                        r[x][y]=(short)negro;
                               if (valor >= umbral)
                                        r[x][y]=(short)blanco;
```

```
                    g[x][y] = r[x][y];
                    b[x][y] = r[x][y];
                }
            }
        short2Image();
    }
```

Los métodos mencionados previamente fueron implementados distribuidamente en nuestra arquitectura y fueron evaluados con las diferentes estrategias de balanceo de carga que hemos venido describiendo. La Figura 7.6 muestra el tiempo de respuesta total cuando evaluamos 1260, 2520, 3780, 5040 y 6300 imágenes heterogéneas, a cada imagen se le aplicó cada uno de los operadores individuales (negativo, escala de grises, la potencia para abrillantar u oscurecer una imagen y la binarizada) [Lyon 99, González 95 y Pajares 04] para transformar el valor de cada píxel de la imagen de entrada y generar una nueva imagen de salida con los píxeles transformados.

Como podemos observar en la Figura 7.6, las dos estrategias que no checan el estado en tiempo de ejecución (Round-Robin y Aleatorio) no son una buena opción para este tipo de aplicaciones. También observamos que de las tres estrategias que checan el estado de ejecución (adaptativas) la que peores resultados genera es la estrategia Umbral aún si la comparamos con las estrategias no-adaptativas pero a medida que se incrementa la carga la estrategia Umbral comienza a ganar terreno en comparación con las estrategias no-adaptativas. Por otra parte la estrategia de la CMC y la Genética (estrategias adaptativas) cuando la carga es ligera pueden ser una buena elección, pero a medida que incrementamos la carga, la estrategia genética se convierte en la mejor opción.

Tiempo Total (Min)					
No. de imágenes	Genética	Aleatoria	Round Robin	CMC	Umbral
1260	9.045	13.060	13.079	10.017	15.067
2520	16.025	25.006	19.067	19.008	23.051
3780	34.059	40.021	41.008	35.019	40.068
5040	36.113	45.012	49.018	39.011	48.032
6300	45.067	66.010	56.011	54.016	62.046

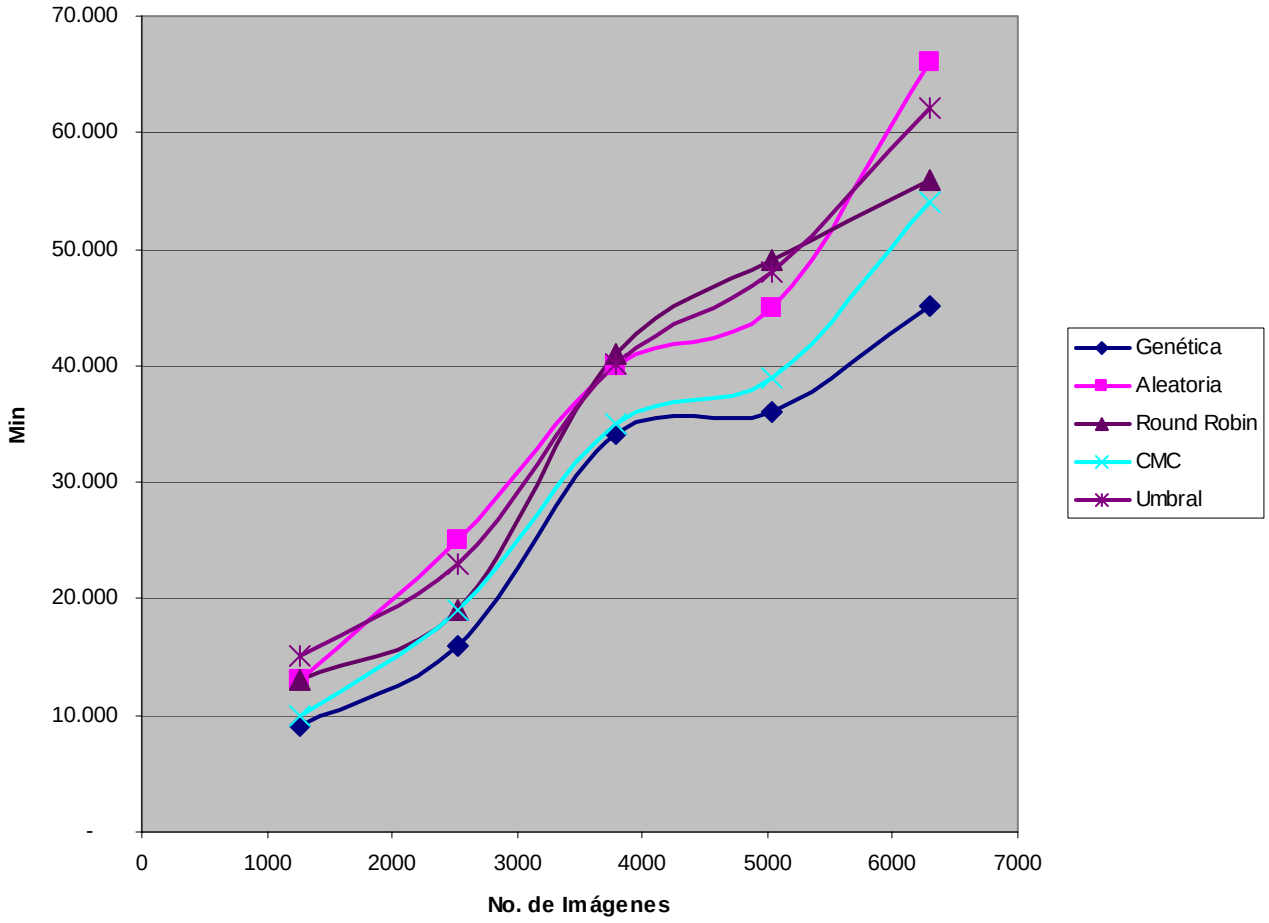


Figura 7.6 Tiempo Total (Balanceo Dinámico de Carga Heterogéneo en el Procesamiento Distribuido de Imágenes).

La utilización de recursos la podemos observar en la Figura 7.7, como observamos en la Figura, las estrategias CMC y Genética son los que mejor aprovechan los recursos y esto es obvio ya que son los que mejores tiempos generan. Si nos apoyamos en los resultados previos podemos decir que el algoritmo Genético genera los mejores resultados cuando la carga de los clientes es elevada.

Rendimiento del Sistema (Mb/Min)					
No. de imágenes	Genética	Aleatoria	Round Robin	CMC	Umbral
1260	27.640	19.142	19.114	24.957	16.593
2520	28.882	21.995	21.156	26.250	20.945
3780	30.425	23.488	23.947	28.700	24.423
5040	33.229	26.660	25.481	30.761	27.253
6300	35.502	24.239	28.566	31.787	29.621

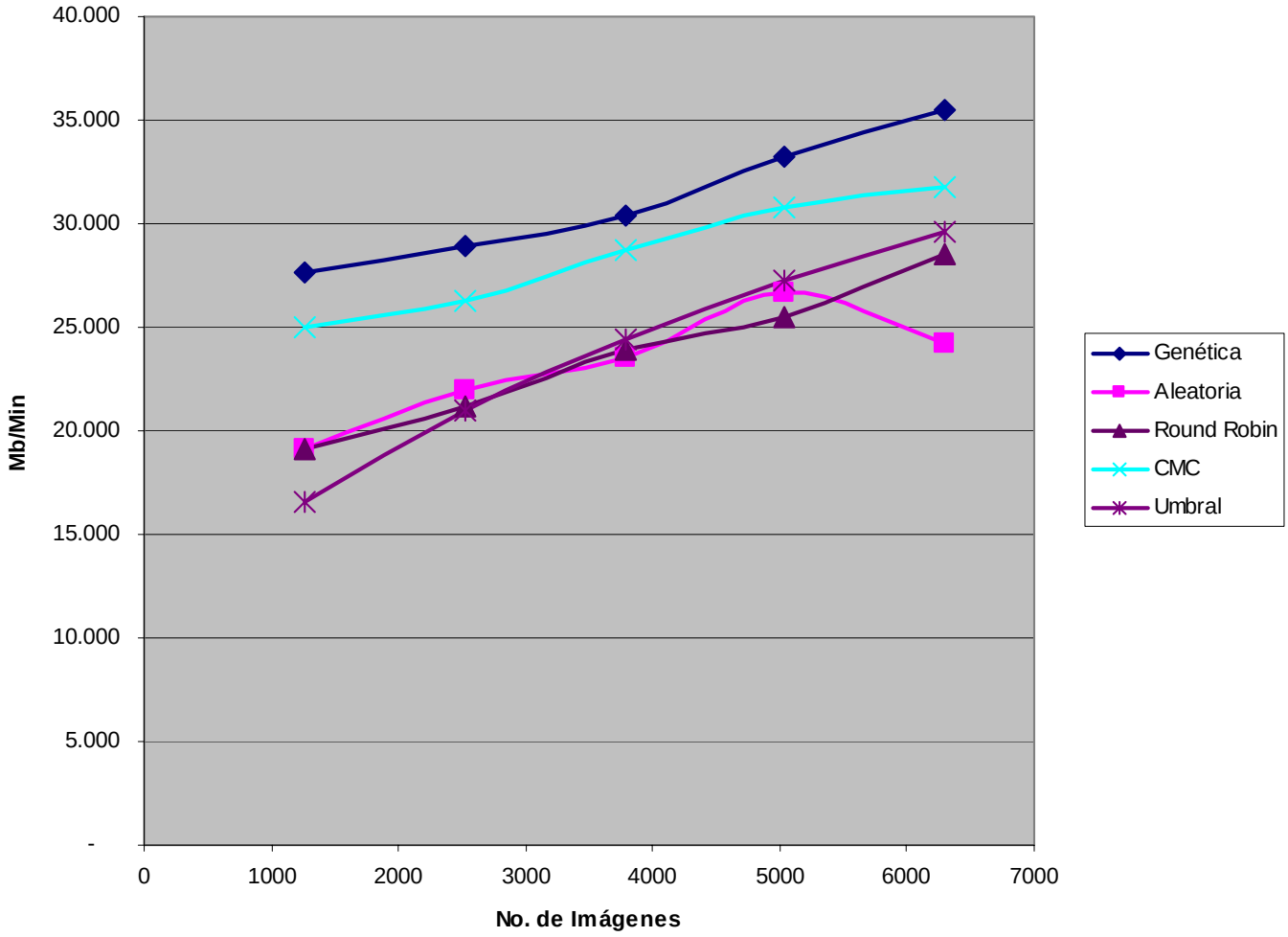


Figura 7.7 Rendimiento del Sistema (Balanceo Dinámico de Carga Heterogéneo en el Procesamiento Distribuido de Imágenes).

7.3 Adelgazamiento de una Imagen Usando el Algoritmo de Zhang y Suen (Esqueleto).

El algoritmo de Zhan y Suen [Zhang], usa un algoritmo basado en dos pasadas a través de la imagen. En cada pasada, una ventana de 3*3 extrae píxeles de la imagen de entrada. La idea básica es que en la

primer pasada obtengamos las condiciones que permitirán el borrado de un píxel (por ejemplo, colocar un píxel de 255 a 0). En nuestra implementación usamos el plano de color rojo como la entrada y el plano de color verde como el almacenamiento temporal. Si un píxel en el arreglo de color rojo es obtenido para conocer el criterio de borrado, marcamos este colocando un 1 en el arreglo verde. Después de obtener el arreglo rojo, obtenemos el arreglo verde. En cualquier parte que el arreglo de color verde no sea un cero borramos el píxel del arreglo de color rojo.

Este procedimiento lo podemos ver implementado en el método *deleteFlagedPoints*:

```
public void deleteFlagedPoints() {
                for ( int x = 1; x < width-1; x++)
                        for ( int y = 1; y < height-1; y++)
                                if (g[x][y] != 0)
                                    r[x][y] = 0;
   }
```

Después de cada pasada a través de la imagen, verificamos si hemos terminado. En el método *skeleton*, invocamos una pasada del algoritmo de adelgazamiento Zhang-Suen. La pasada retorna *true* si un píxel es alterado; en otro caso la pasada retorna *false*. Si en cada pasada retornamos *false*, entonces hemos terminado. Para determinar si la pasada es la primera o la segunda, usamos un argumento como *boolean.* Si el *boolean* es *true,* entonces estamos en la primer pasada. Las condiciones para adelgazar un píxel cambian ligeramente en la segunda pasada.

El método *skeleton* es implementado como sigue:

```
public void skeleton() {
        while (
                skeletonRedPassSuen(true)&&
                skeletonRedPassSuen(false)) {
        }
       copyRedToGreenAndBlue();
```

```
        short2Image();
    }
```

La respuesta es dejada en al arreglo rojo, y esta es copiada al arreglo verde y azul al final del método.

En la Figura 7.8 observamos la localización de los píxeles en una ventana 3*3.

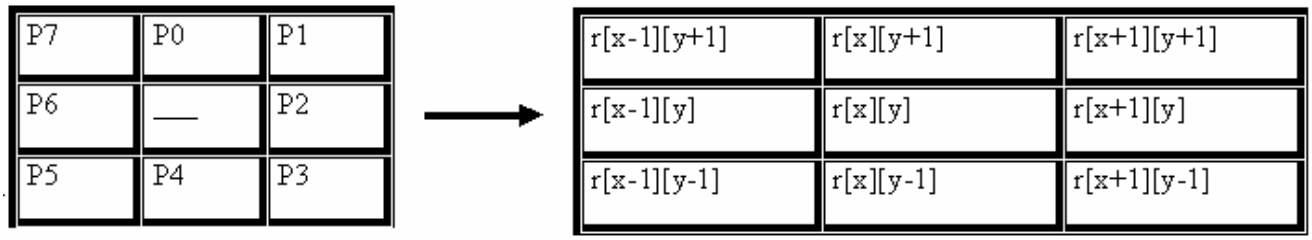


*Figura 7.8 Localización de Píxeles en una Ventana de 3*3.*

Un arreglo *boolean* de ocho elementos es usado para mantener la pista de los píxeles que no son cero en una ventana 3*3. Si el origen de la ventana (r[x][y]) es cero, entonces procedemos al siguiente píxel usando la declaración *continue* de java. Durante esta pasada, no aseguramos que la salida sea un cero (usamos el plano rojo para esto). Durante la primera pasada brincamos al siguiente píxel si alguna de las siguientes condiciones ocurre.

1.- Si el origen r[x][y] = 0.
2.- El número de vecinos es menor que 2.
3.- El número de vecinos es mayor que 6.
4.- El número de transiciones 0 – 1 es igual a 1.
5a.- Si (p[0] && p[2] && p[4]) es verdadero.
6a.- Si (p[2] && p[4] && p[6]) es verdadero.

Durante la segunda pasada, sustituimos 5ª y 6ª con

5b.- Si (p[0] && p[2] && p[6]) es verdadero.
5b.- Si (p[0] && p[4] && p[6]) es verdadero.

La regla 1 asegura que el píxel tenga un valor de procesamiento. La regla dos asegura que se separen los puntos y los tips del esqueleto que no serán adelgazados. La regla tres asegura que r[x][y] esté sobre el límite de la ventana 3*3, mostrada en la Figura 7.8. La regla cuatro asegura que el esqueleto no llegue a fragmentarse. Las reglas 5 y 6 mantienen conectadas las líneas que son de dos píxeles de ancho.

Cuando las seis reglas son falsas, marcamos el píxel para borrar e incrementamos un contador indicando cuantas marcas fueron hechas. Para obtener la condición 1, usamos:

```
if ( g[x][y] != 0 ) continue;
```

Donde la operación *continue* permite al píxel ser brincado. Si esta trivial condición es omitida, entonces el arreglo *boolean* es colocado a *true* cuando los píxeles sean locales. De esta forma, por cada píxel en la imagen, la ventana 3*3 es mapeada dentro de un arreglo 1*8.

```
p[0] = r[x][y+1]   != 0;
p[1] = r[x+1][y+1] != 0;
p[2] = r[x+1][y]   != 0;
p[3] = r[x+1][y-1] != 0;
p[4] = r[x][y-1]   != 0;
p[5] = r[x-1][y-1] != 0;
p[6] = r[x-1][y]   != 0;
p[7] = r[x-1][y+1] != 0;
```

Conocemos de las condiciones dos y tres, que el número de vecinos está entre dos y seis. Así, si el número de vecinos es menor que dos o mayor que seis, continuamos al siguiente píxel:

```
int n = numberOfNeighbors(p);
if ((n < 2) || (n > 6)) continue;
```

La condición 4 requiere que contemos el número de transiciones cero-uno:

```
    private int numberOf01Transitions(boolean p[]) {
        int n=0;
        if ((!p[0]) && p[1]) n++;
        if ((!p[1]) && p[2]) n++;
        if ((!p[2]) && p[3]) n++;
        if ((!p[3]) && p[4]) n++;
        if ((!p[4]) && p[5]) n++;
        if ((!p[5]) && p[6]) n++;
        if ((!p[6]) && p[7]) n++;
        if ((!p[7]) && p[0]) n++;
    return n;
}
```

Si el número de transiciones no es igual a uno, continuamos al siguiente píxel:

```
    if (numberOf01Transitions(p) != 1) continue;
```

Las condiciones 5a, 6a y 5b, 6b son implementadas, usando.

```
    if ( firstPass ) {
        if (( p[0] && p[2] && p[4] ) ) continue;
        if (( p[2] && p[4] && p[6] ))  continue;
        g[x][y] = 255;
        c++;
        }
    else {
       if (( p[0] && p[2] && p[6] ))          continue;
        if (( p[0] && p[4] && p[6] ))         continue;
        g[x][y] = 255;
        c++;
    }
```

El código completo es como sigue:

```
public boolean skeletonRedPassSuen(
                boolean firstPass) {
        boolean p[]=new boolean[8];
        short c = 0;
         for(int x = 1; x < width-1; x++) {
                for(int y = 1; y < height-1; y++) {
                                g[x][y] = 0;
                                if (r[x][y] == 0) continue;
                                p[0] = r[x][y+1]  != 0;
                                p[1] = r[x+1][y+1]!= 0;
                                p[2] = r[x+1][y]  != 0;
                                p[3] = r[x+1][y-1]!= 0;
                                p[4] = r[x][y-1]  != 0;
                                p[5] = r[x-1][y-1]!= 0;
                                p[6] = r[x-1][y]  != 0;
                                p[7] = r[x-1][y+1]!= 0;
                                int n = numberOfNeighbors(p);
                                if ((n < 2) || (n > 6))
                                                continue;
                                if (numberOf01Transitions(p) != 1)
                                                continue;
                                 if (firstPass) {
                                        if ((p[0] && p[2] && p[4]) )
                                                continue;
                                        if ((p[2] && p[4] && p[6]))
                                                continue;
                                        g[x][y] = 255;
                                        c++;
                                        }
```

```
                    else {
                        if ((p[0] && p[2] && p[6]))
                                continue;
                        if ((p[0] && p[4] && p[6]))
                                continue;
                        g[x][y] = 255;
                        c++;

                    }
                }
            }
            if (c == 0) return false;
            deleteFlagedPoints();
            return true;
            }
```

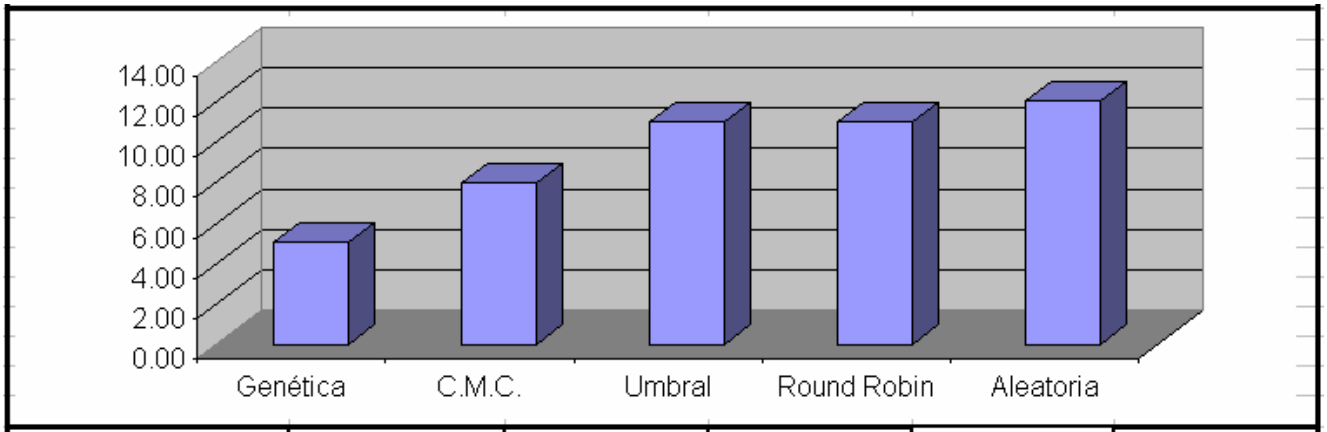
Para invocar las dos pasadas del algoritmo, alteramos los argumentos entre *true* y *false*. Si cada pasada retorna *false*, entonces ningún píxel fue alterado, y el procesamiento es completado.

```
public void skeleton() {
    while (
        skeletonRedPassSuen(true)&&
        skeletonRedPassSuen(false)) {
    }
      copyRedToGreenAndBlue();
  }
```

Figura 7.9 Aplicación del Algoritmo Zhan y Suen para Adelgazar una Imagen [Lyon 99].

La Figura 7.9 muestra el algoritmo de adelgazamiento Zhan y Suen aplicado para adelgazar una imagen. El adelgazamiento se aplica a un *font* de tipo *Times New Roman,* como podemos observar en la figura, el *font* es adelgazado a un solo píxel.

La Figura 7.10 muestra el tiempo de respuesta global que genera el algoritmo de Zhan y Suen cuando se adelgazaron un total de 180 imágenes heterogéneamente. El adelgazamiento asume que el ruido ha sido filtrado en cada imagen. El tipo de filtro usado dependerá de la composición del ruido. Después de que se ha filtrado el ruido, se binariza la imagen, posteriormente se saca el negativo de la imagen, una vez que se invierte la imagen se aplica el algoritmo de Zhan y Suen y nuevamente la imagen resultante se vuelve a invertir. Cabe hacer mención que no siempre se aplica el mismo preprocesamiento a todas las imágenes, la aplicación del tipo de preprocesamiento dependerá de la aplicación que se desarrolle.



Estrategia	Genética	C.M.C.	Umbral	Round Robin	Aleatoria
Tiempo (MIN.)	5.04	8.01	11.00	11.00	12.02

Figura 7.10 Tiempo Total (Algoritmo de Zhan y Suen). Balanceo Dinámico de Carga Heterogéneo en el Procesamiento Distribuido de Imágenes.

7.4 Convolución.

En procesamiento de imágenes, convolución es una operación tomada entre dos imágenes. Una imagen es típicamente más pequeña que la otra. La imagen más pequeña es llamada el kernel de la convolución. En realidad la convolución es la correlación entre la imagen y el kernel.

La convolución puede detectar bordes, suavizar una imagen, realzarla, etc. Convolución es una técnica primaria para filtrado espacial. Un filtro espacial toma el área alrededor del píxel de una imagen y ejecuta una operación sobre cada uno de los píxeles de tal forma que creamos un nuevo píxel en la imagen de salida. La dimensión del área del píxel de interés esta dada por la máscara de convolución y forma una ventana de convolución. La ventana de convolución es movida y esta se centra en cada píxel en la imagen de entrada. Después de centrar el píxel, este es calculado usando la suma de productos.

Definimos la convolución discreta de una dimensión como la correlación entre dos señales que han sido invertidas en el tiempo. En el dominio continuo, escribimos:

$$h(x) = f(x)\cdot g(x) = \int_{-\infty}^{\infty} f(u)\cdot g(x-u)\,du \tag{7.6}$$

donde f es la señal de entrada, u es una variable tonta de integración, y g es llamado el kernel de la convolución. Otros nombres del kernel incluyen ventana del kernel, máscara, ventana, filtro, etc. Nosotros usamos el término kernel. La ecuación 7.6 desliza el kernel a través de la señal de entrada. Esta multiplica la señal de entrada con el kernel y suma el resultado. Típicamente 7.6 esta dada por:

$$\mathrm{h(x) = f(x)\cdot g(x) = \sum_{menus=\infty}^{\infty} f(u)\cdot g(x-u)} \tag{7.7}$$

Cuando la señal de entrada y el kernel se extienden a infinito y el kernel tiene una región finita de soporte, decimos que es una convolución lineal. Una función que tiene una región finita se dice que tiene un soporte compacto. Cuando la señal de entrada y el kernel tienen secuencias periódicas con el mismo periodo, entonces la convolución se dice que es una convolución cíclica (algunas ocasiones esta es llamada convolución circular). Regularmente el kernel es más pequeño que la señal de entrada, por lo tanto hacemos repeticiones de tal forma que obtengamos la misma longitud que la señal de entrada. La convolución resultante es una secuencia que es de la misma longitud que la señal de entrada. La convolución es también periódica con el mismo periodo que la señal de entrada.

La convolución continua es definida en dos dimensiones como:

$$h(x,y) = f\cdot g = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(u,v)\cdot g(x-u, y-v)\,du\,dv \tag{7.8}$$

En el dominio discreto (7.8) debe ser escrita como:

$$h(x,y) = f\cdot g = \sum_{u=0}^{u_{max}-1}\sum_{v=0}^{v_{max}-1} f(u,v)\,g[(x-u),(y-v)] \tag{7.9}$$

donde g es el kernel periódico de la convolución , f es la imagen de entrada periódica en dos dimensiones, h es la convolución de salida cíclica en dos dimensiones, u_{max} es el ancho del kernel y v_{max} es el alto del kernel.

7.5 Transformaciones de Vecindad: Suavizado.

La generación de un nuevo píxel en una nueva imagen será una función que depende del valor de cada píxel en su localización individual, o bien de los valores de los píxeles en la vecindad de un píxel dado, como se indica en la Figura 7.11.

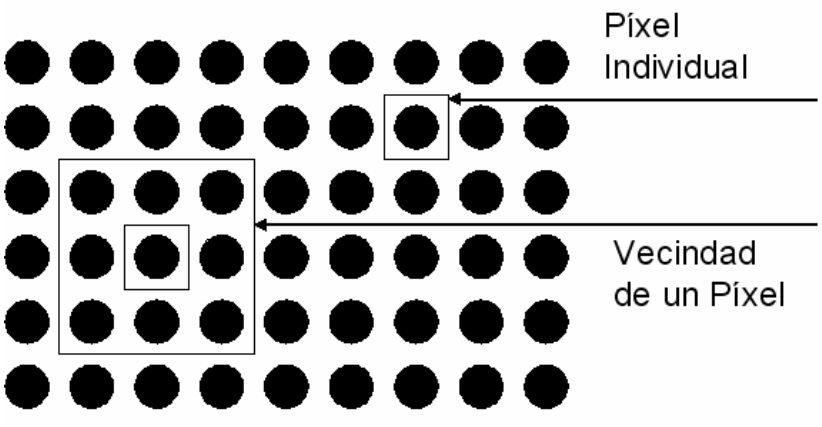


Figura 7.11 Vecindad de Píxeles.

Dentro de la categoría de operaciones de vecindad se incluyen las operaciones de filtrado. Las operaciones de filtrado tienen la particularidad de eliminar un determinado rango de frecuencias de las imágenes.

7.5.1 Filtros.

Las operaciones de filtrado basan su operatividad en la convolución de la imagen utilizando el denominado kernel de convolución. Cabe distinguir dos tipos de filtros: paso altas y paso bajas, que en el contexto de la teoría de señales, supone que los primeros dejan pasar las altas frecuencias de la señal y los segundos las bajas. En el caso de las imágenes nos referimos a frecuencias espaciales. De esta forma las altas frecuencias se asocian a cambios bruscos de intensidad en pequeños intervalos

espaciales, es decir bordes, mientras que las bajas frecuencias se refieren a cambios lentos en la intensidad. Por lo tanto se deduce que los filtros paso bajas al no dejar pasar las altas frecuencias (bordes), atenuará éstas. El resultado es un desenfoque o desdibujado de los bordes. No obstante considerando el ruido de la imagen como un elemento no deseado y también asociado a las altas frecuencias, este tipo de filtros actuará muy bien de cara a eliminar píxeles de ruido existentes en la imagen. Se entiende por ruido aquellos píxeles que tienen valores muy diferentes a sus vecinos en regiones homogéneas. El ruido procede de funcionamientos erróneos o imperfecciones de algunos elementos del dispositivo de captura de la imagen, tales como celdas en mal funcionamiento, o también debido a las impurezas, tales como partículas de polvo, suciedad, imperfecciones, etc., en la óptica del sistema de captura. Por otro lado, volviendo a la funcionalidad de los filtros, los denominados paso altas, al dejar pasar las altas frecuencias (bordes), producen un realzado de los bordes de la imagen atenuando la zona donde no hay cambios bruscos de intensidad.

7.5.2 Filtros Paso Bajas.

Un tipo de filtro paso bajas que puede ser ejecutado en una imagen es llamado vecindad promedio (también llamado filtro de suavizado). Típicamente tal filtro consiste de la suma de los pesos de todos los píxeles en la vecindad, dividido por algún factor de escala. El factor de escala intenta que todos los pesos de los elementos en el kernel sumen uno. La Figura 7.12 muestra la imagen original del mandril seguida de 3 imágenes que resultan de hacer la convolución con diferentes kernels 3*3. Los kernels son llamados: promedio, lp3 y Gaussian.

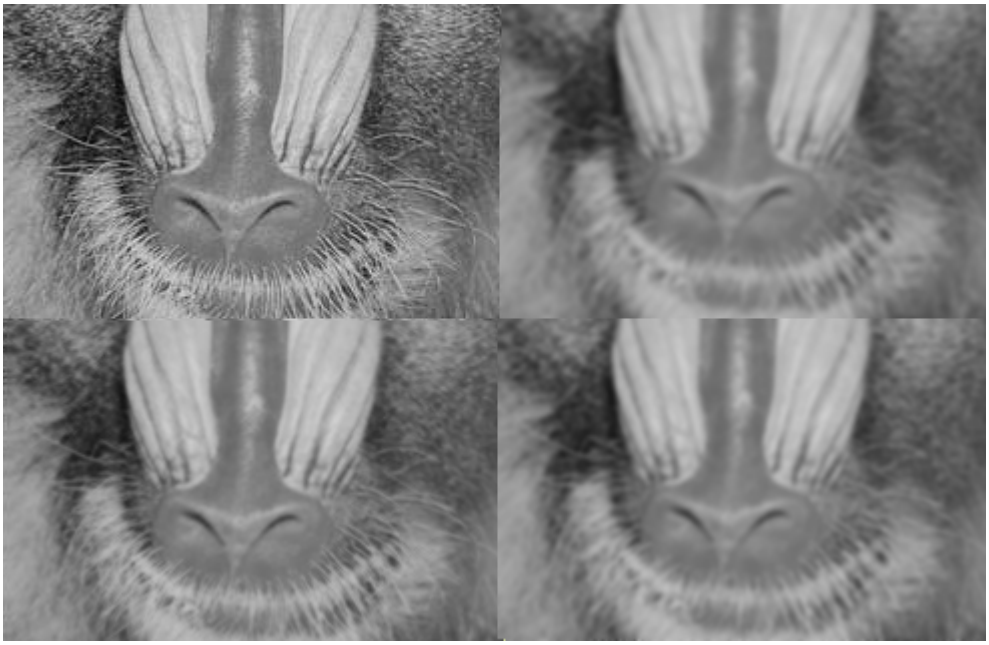


Figura 7.12 Mandril, Average(), Lp3() y Gaussian() [Lyon 99].

Filtro Promedio (Average()). Es un ejemplo de un filtro paso bajas. El kernel del filtros paso bajas es caracterizado por que los elementos son positivos y suman un uno. En el caso de *average()*, los elementos suman 9 y por lo tanto dividimos por 9, usando *Mat.scale(k, 1/9.0).*

```
public void average() {
        float k[][] = {
                {1, 1, 1},
                {1, 1, 1},
                {1, 1, 1}
                };
                Mat.scale(k,1/9.0);
                convolve(k);
 }
```

Filtro Lp3. Notamos en la Figura 7.12, que la imagen inferior izquierda (*Lp3()*) aparece menos suavizada en comparación con la imagen superior derecha. La razón es que el píxel del centro en el kernel tiene más peso que los píxeles que están a su alrededor. Este filtro ayuda a remover el ruido de

la imagen, pero también tiende a borrar los bordes. El código del kernel *Lp3()* es como sigue:

```
public void Lp3() {
        float k[][] = {
                { 1,  1,   1},
                { 1,  12, 1},
                { 1,  1,   1}
                };
                Mat.scale(k,1/20.0);
                convolve(k);
        }
```

Filtro Gaussian. En 2D, la densidad Gaussian esta dada por.

$$\left(gaussian\left(x, y, x_c, y_c, \sigma\right) = \frac{1}{2\pi\sigma} \right) e^{\frac{-\left[\left(x-x_c\right)^2-\left(y-y_c\right)^2\right]}{2\cdot\sigma^2}} \tag{7.10}$$

donde σ es la desviación estándar. El máximo valor de (7.10) ocurre en $x = x_c, y = y_c$ y esta dado por:

$$g_{max} = \frac{1}{2\pi\cdot\sigma^2} \tag{7.11}$$

El código del kernel Gaussian es como sigue:

```
public void gauss3() {
        float k[][] = {
                {1, 2,  1},
                {2, 4,  2},
                {1, 2,  1}
                };
```

```
        Mat.scale(k,1/16.0);
        convolve(k);
}
```

7.5.3 Filtro Paso Altas.

Los filtros paso altas (también conocidos afilado de bordes) dejan pasar las altas frecuencias de la señal y dejan fuera los pequeños detalles de la imagen. Pruebas clínicas han demostrado que los filtros paso altas pueden ser evaluados más subjetivamente en imágenes no distorsionadas [Lyon 99]. La convolución directa puede ser usada para ejecutar un filtro paso altas usando un kernel en el cual sus elementos tengan ambos signos: positivos y negativos. La Figura 7.13 muestra la imagen original del mandril seguida de 3 imágenes que resultan de hacer la convolución con diferentes kernels 3*3. Los kernels son llamados: *hp1(), hp2() y hp3().*

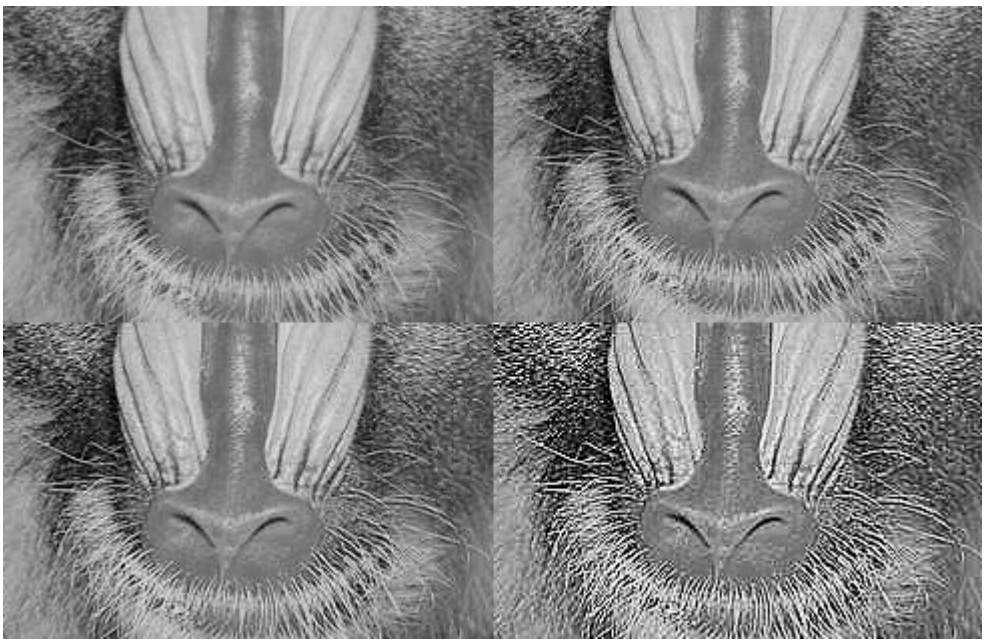


Figura 7.13 Mandril, hp1(), hp2() y hp3() [Lyon 99].

El código de los kernels *hp1(), hp2() y hp3()* es como sigue:

```
public void hp1() {
```

```
 float k[][] = {
        { 0, -1,  0},
        {-1, 10, -1},
        { 0, -1,  0}
        };
Mat.normalize(k);
convolve(k);
}
public void hp2() {
 float k[][] = {
        { 0, -1,  0},
        {-1,  8, -1},
        { 0, -1,  0}
        };
        Mat.normalize(k);
        convolve(k);
  }
public void hp3() {
 float k[][] = {
        { 0, -1,  0},
        {-1,  5, -1},
        { 0, -1,  0}
        };
        Mat.normalize(k);
        convolve(k);
  }
```

7.6 Convolución Basada en Detección de Bordes.

La detección de bordes es parte del procesamiento de imágenes que es usado para delinear los límites de los objetos en la imagen [Lyon 99]. La extracción es útil y el realzado de bordes es el problema

principal en el procesamiento de imágenes.

7.6.1 Operador Prewitt.

Una forma de detectar un borde es calcular el gradiente de una imagen y entonces trasladar el gradiente en una cantidad escalar que pueda ser usada por alguna función de decisión. El gradiente es procesado como la derivada parcial en cada una de las direcciones *x e y*. El resultado de procesar el gradiente es un vector.

$$\Delta f(x,y) = \frac{\delta f}{\delta x} \cdot i + \frac{\delta f}{\delta y} \cdot j \tag{7.12}$$

El valor absoluto del gradiente es un escalar dado por la raíz cuadrada de la suma de los cuadrados:

$$\left|\Delta f(x,y)\right| = \sqrt{\left(\frac{\delta f}{\delta x}\right)^2 + \left(\frac{\delta f}{y}\right)^2} \tag{7.13}$$

El operador de Prewitt es otra forma de detectar bordes, con la detección de bordes Prewitt, nos enfrentamos a la tarea de realizar dos convoluciones por cada píxel. Generalmente este se aplica a la detección de bordes verticales y horizontales. Tomamos la raíz cuadrada de la suma de los cuadrados por cada uno de los resultados de las convoluciones (7.13) y usamos el resultado en la imagen de salida. Los dos kernels para la detección de bordes de Prewitt horizontal y vertical están dados por el método siguiente:

```
public void prewitt() {
        float k1[][] = {
                { 1,   0, -1},
                { 1,   0, -1},
                { 1,   0, -1}
                };
```

```
            float k2[][] = {
                    {-1,  -1, -1},
                    { 0,   0,  0},
                    { 1,   1,  1}
                    };
            Mat.scale(k1,1/3.0);
            Mat.scale(k2,1/3.0);
            templateEdge(k1,k2);
    }
```

El procesamiento central en Prewitt es ejecutado en el método *templateEdge*. Comenzamos el proceso convirtiendo la imagen a escala de grises. Esto lo realizamos colocando el promedio de los planos, rojo, verde y azul en el plano rojo. Entonces ejecutamos una convolución entre el kernel k1 y el plano rojo, y colocamos la respuesta en el plano verde. La convolución entre k2 y el plano rojo, es colocada en el plano azul. La raíz cuadrada de la suma de los cuadrados en los planos verde y azul es calculada y posteriormente colocada en los tres planos. El método *templateEdge* es el siguiente:

```
public void templateEdge(float k1[][], float k2[][]) {
    colorToRed();
    g = convolve(r,k1);
    b = convolve(r,k2);
    short t = 0;
    for (int x=0; x < width; x++)
            for (int y=0; y < height; y++) {
              t = (short)Math.sqrt(g[x][y]*g[x][y]+b[x][y]*b[x][y]);
              r[x][y] = t;
              g[x][y] = r[x][y];
              b[x][y] = r[x][y];
    }
    short2Image();
}
```

```
public void colorToRed() {
  for (int x=0; x < width; x++)
        for (int y=0; y < height; y++)
              r[x][y] = (short)
                ((r[x][y] + g[x][y]  + b[x][y]) / 3);
}
```

Para entender mejor que sucede cuando ejecutamos dos convoluciones, colocamos un kernel 3*3, usando (7.14):

$$\begin{bmatrix} f(x-1)(y+1) & f(x)(y+1) & f(x+1)(y+1) \\ f(x-1)(y) & f(x)(y) & f(x+1)(y) \\ f(x-1)(y-1) & f(x)(y-1) & f(x+1)(y-1) \end{bmatrix} \tag{7.14}$$

Tomamos el kernel k2 y hacemos convolución con (7.14) para obtener:

$$\Delta_x = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} \cdot \begin{bmatrix} f(x-1)(y+1) & f(x)(y+1) & f(x+1)(y+1) \\ f(x-1)(y) & f(x)(y) & f(x+1)(y) \\ f(x-1)(y-1) & f(x)(y-1) & f(x+1)(y-1) \end{bmatrix} \tag{7.15}$$

Expandiendo (7.15) producimos:

$$\begin{aligned} \Delta_x = {} & f(x-1)\cdot(y+1) + f(x-1)\cdot(y) + f(x-1)\cdot(y-1) \ \ldots \\ & + [\,-[\, f(x+1)\cdot(y+1) + f(x+1)\cdot(y) + (x+1)\cdot(y-1)\,]\,] \end{aligned} \tag{7.16}$$

Tomamos el kernel k1 y hacemos convolución con (7.14) para obtener:

$$\Delta_x = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \cdot \begin{bmatrix} f(x-1)(y+1) & f(x)(y+1) & f(x+1)(y+1) \\ f(x-1)(y) & f(x)(y) & f(x+1)(y) \\ f(x-1)(y-1) & f(x)(y-1) & f(x+1)(y-1) \end{bmatrix} \tag{7.17}$$

Expandiendo (7.17) producimos:

$$\Delta_x = f(x-1)\cdot(y-1) + f(x)\cdot(y-1) + f(x+1)\cdot(y-1) \ldots$$
$$+ [\,-[\, f(x-1)\cdot(y+1) + f(x)\cdot(y+1) + f(x+1)\cdot(y+1)\,]\,] \tag{7.18}$$

El kernel Prewitt es escalado a 0.3333, entonces el método *templateEdge* procesa la raíz cuadrada de la suma de los cuadrados:

$$|\Delta f(x,y)| = \sqrt{\left(\frac{\delta f}{3}\right)^2 + \left(\frac{\delta f}{3}\right)^2} \tag{7.19}$$

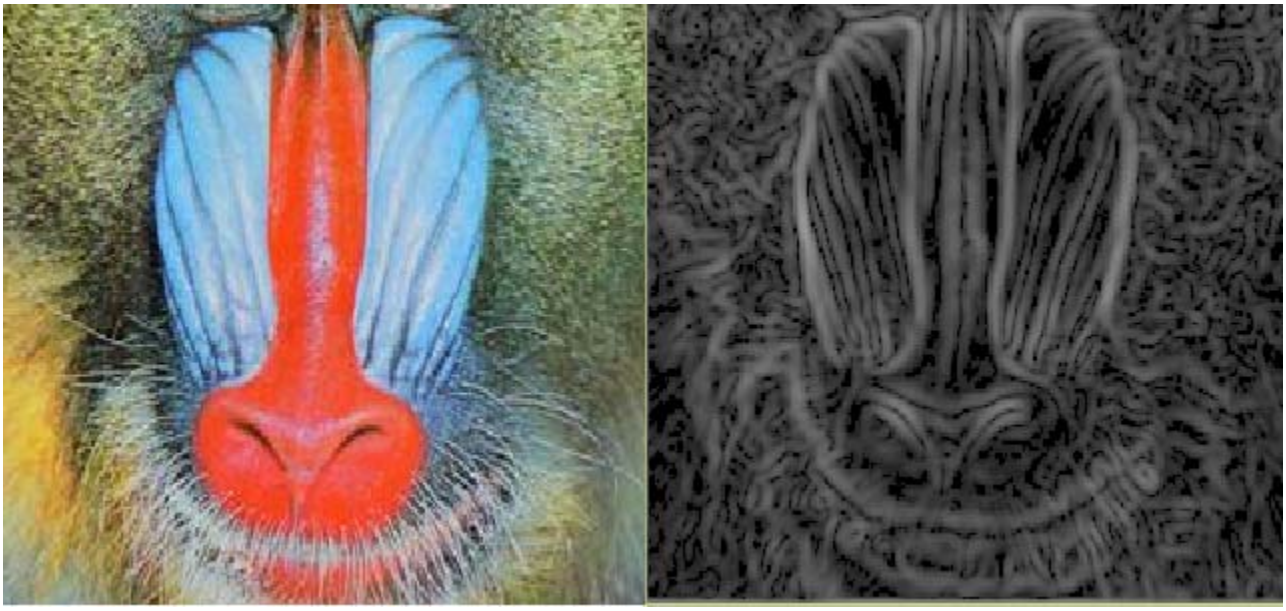
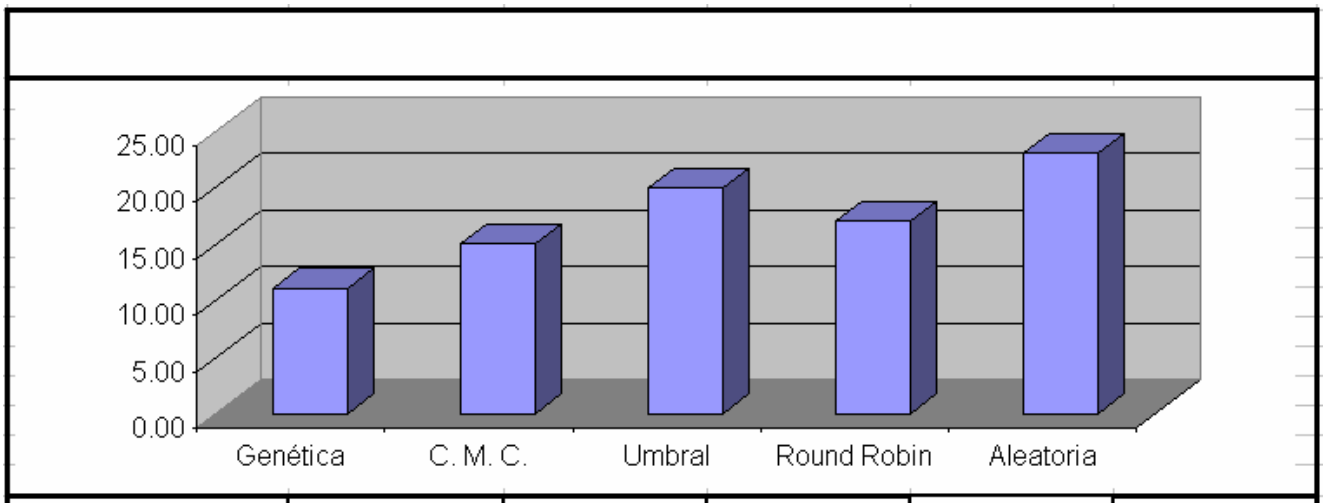


Figura.7.14 Antes y Después de Aplicar un Operador Prewitt [Lyon 99].

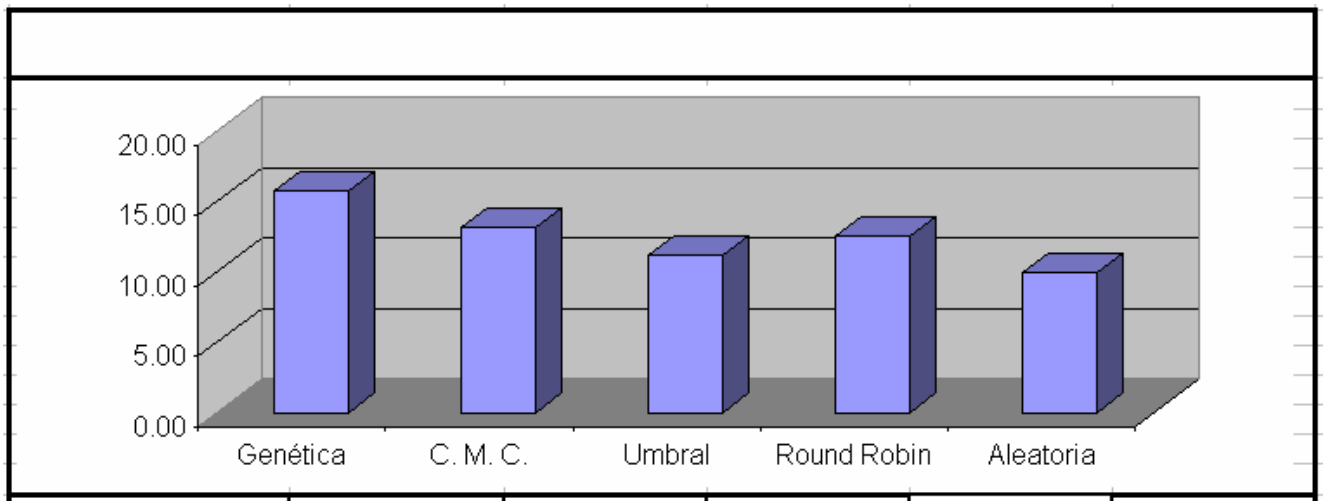
La Figura 7.14 muestra el mandril antes y después de aplicar un operador Prewitt, la imagen fue previamente pre-procesada, aplicamos la escala de grises, enseguida un filtro paso bajas, típicamente el filtro consiste en la suma de los pesos de todos los píxeles en la vecindad, dividido entre algún factor de escala. El factor de escala intenta que todos los elementos de los pesos sumen uno o sea que aplicamos el filtro promedio. Después de realizarle el pre-procesamiento a la imagen aplicamos el operador Prewitt descrito previamente, y posteriormente abrillantamos la imagen 3 veces con el propósito de visualizar con mayor claridad los bordes del mandril.



Estrategia	Genética	C.M.C.	Umbral	Round Robin	Aleatoria
Tiempo(Min.)	11.11	15.02	20.02	17.02	23.03

Figura 7.15 Tiempo de Respuesta (Operador Prewitt). Balanceo Dinámico de Carga Heterogéneo en el Procesamiento Distribuido de Imágenes.

La Figura 7.15 muestra el comportamiento de las estrategias cuando aplicamos el operador Prewitt a 800 imágenes heterogéneas en diferentes formatos (GIF y JPEG), como podemos observar en la figura, la estrategia genética, mejora el tiempo de respuesta en casi cuatro minutos en comparación con la estrategia de la Cola Más Corta (CMC) y en casi 9 minutos si la comparamos con la otra estrategia adaptativa (Umbral). También observamos como la estrategia genética genera tiempos de respuesta bastante considerables en comparación con las estrategias no-adaptaivas (Round_Robin y Aleatoria). La Figura 7.16 muestra el aprovechamiento de recursos de las estrategias, como podemos observar en la figura, la estrategia genética es la que mejor aprovechamiento de recursos tiene y esto es correcto ya que esta estrategia es la que mejores tiempos de respuesta genera.



Estrategia	Genética	C.M.C.	Umbral	Round Robin	Aleatoria
Rendimiento(Mb/Min	15.69	13.18	11.22	12.49	9.93

Figura 7.16 Rendimiento del Sistema (Operador Prewitt). Balanceo Dinámico de Carga Heterogéneo en el Procesamiento Distribuido de Imágenes.

7.6.2 Operador Laplaceano.

Una manera de formular un kernel para diseñar un filtro de detección de bordes es usando un operador Laplace. P. S. Laplace (1749-1827) fue un matemático francés quien contribuyo al campo de la astronomía, probabilidad y mecánica [Lyon 99]. Una de sus contribuciones, llamada la transformada de Laplace, es típicamente estudiado en un primer curso de ecuaciones diferenciales. El operador Laplace (también llamado laplaciano), es totalmente diferente de la transformada Laplace. El operador Laplaceano, en dos dimensiones, es una derivada parcial de segundo orden, tomada con respecto a cada una de las direcciones ortogonales. Esta también es llamada la divergencia del gradiente y esta dado por:

$$\Delta^2 f(x,y) = \frac{\delta^2 f}{\delta x^2} + \frac{\delta^2 \cdot f}{\delta y^2} \tag{7.20}$$

Estamos interesados en el operador Laplacian por que este es cero cuando la razón de intensidad cambia en el plano de la imagen. Por ejemplo, supongamos que:

$$f(x, y) = x^2 + y^2 \tag{7.21}$$

Para procesar (7.20) necesitamos tomar la segunda derivada parcial con respecto a cada coordenada, para obtener:

$$\frac{\delta^2 f}{\delta x^2} = \frac{\delta^2 \cdot f}{\delta y^2} = 2 \tag{7.22}$$

Sustituyendo (7.22) en (7.20) obtenemos:

$$\Delta^2 f(x, y) = 4 \tag{7.23}$$

Ahora afirmamos que la primera y segunda derivada de una función uniforme pudiera ser aproximada a una convolución sobre tres ejemplos espaciados. Esta aseveración forma las bases de los detectores de bordes 3*3. La convolución es usada para crear la equivalencia computacional de las series de Taylor para aproximarse a la derivada [Lyon 99].

Para entender como aplicar el operador Laplaciano a la imagen discreta, considere una ventana de 3*3 centrada sobre el píxel localizado en *f[x][y]*, dado por:

$$\begin{bmatrix} f(x-1)(y+1) & f(x)(y+1) & f(x+1)(y+1) \\ f(x-1)(y) & f(x)(y) & f(x+1)(y) \\ f(x-1)(y-1) & f(x)(y-1) & f(x+1)(y-1) \end{bmatrix} \tag{7.24}$$

Aproximamos la primera derivada parcial en la dirección *x* (de la derecha al centro) como:

$$\frac{\delta f_r}{\delta x} = f(x+1)(y) - f(x) \cdot (y) \tag{7.25}$$

Podemos también aproximar la primera derivada parcial en la dirección *x* (del centro a la izquierda)

como

$$\frac{\delta f_l}{\delta x} = f(x)(y) - f(x-1)\cdot(y) \tag{7.26}$$

La segunda derivada parcial con respecto a *x*, es aproximada restando (7.25) de (7.26) para obtener:

$$\frac{\delta^2 f}{\delta x^2} = \frac{\delta f_l}{\delta x} - \frac{\delta f_r}{\delta x} = 2\, f(x)(y) - f(x-1)(y) - f(x+1)(y) \tag{7.27}$$

Similarmente, podemos aproximar la primera derivada parcial en la dirección *y* (de arriba hacia el centro) usando:

$$\frac{\delta f_t}{\delta y} = f(x)(y+1) - f(x)\cdot(y) \tag{7.28}$$

También aproximamos la primera derivada parcial en la dirección *y* (del centro hacia abajo) con:

$$\frac{\delta f_b}{\delta y} = f(x)(y) - f(x)\cdot(y-1) \tag{7.29}$$

De la misma forma que en la dirección *x*, formulamos la segunda derivada parcial, con respecto a *y*, restando (7.28) de (7.29) para obtener:

$$\frac{\delta^2 f}{\delta y^2} = \frac{\delta f_b}{\delta x} - \frac{\delta f_t}{\delta x} = 2\, f(x)(y) - f(x)(y-1) - f(x)(y+1) \tag{7.30}$$

Ahora sustituimos (7.30) y (7.28) en (7.20) para obtener la aproximación del operador Laplaciano sobre un kernel 3*3.

$$\Delta^2 f(x,y) = 2\cdot f(x)\cdot(y) - f(x-1)\cdot(y) - f(x+1)\cdot f(y) + 2\, f(x)\cdot(y) - f(x)\cdot(y-1) - f(x)\cdot(y+1) \quad (7.31)$$

Que simplifica a:

$$\Delta^2 f(x,y) = 4\cdot f(x)\cdot(y) - f(x-1)\cdot(y) - f(x+1)\cdot(y) - f(x)\cdot(y-1) - f(x)\cdot(y+1) \quad (7.32)$$

La aproximación de (7.32) es procesada por convolución con un kernel Laplaciano 3*3, dado por:

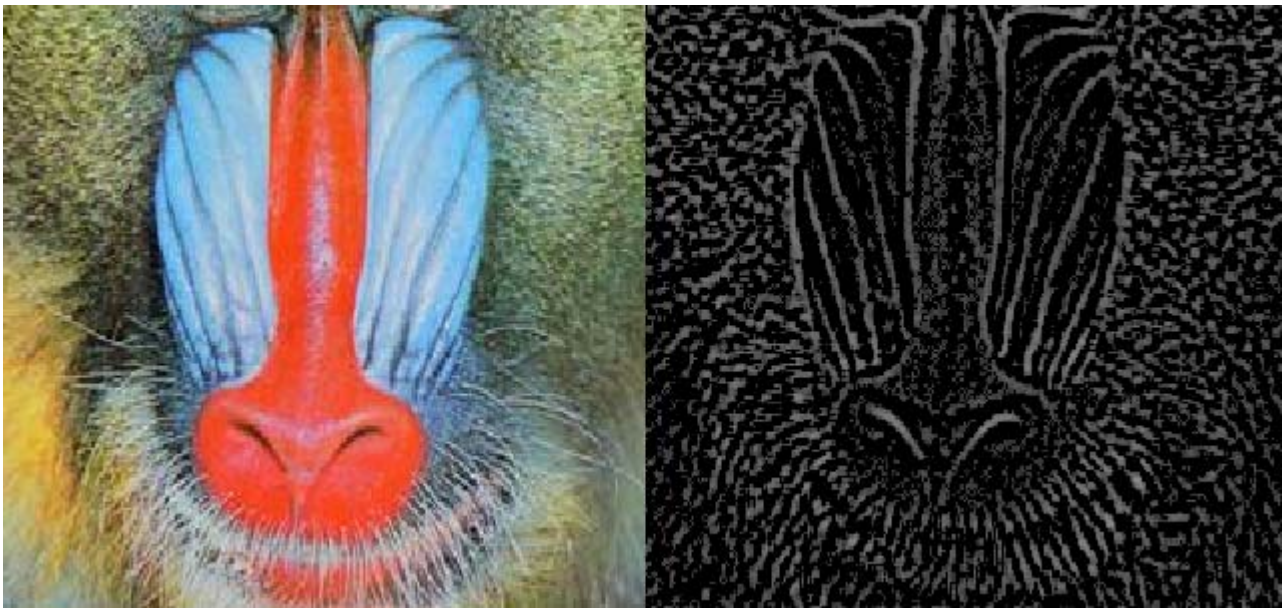
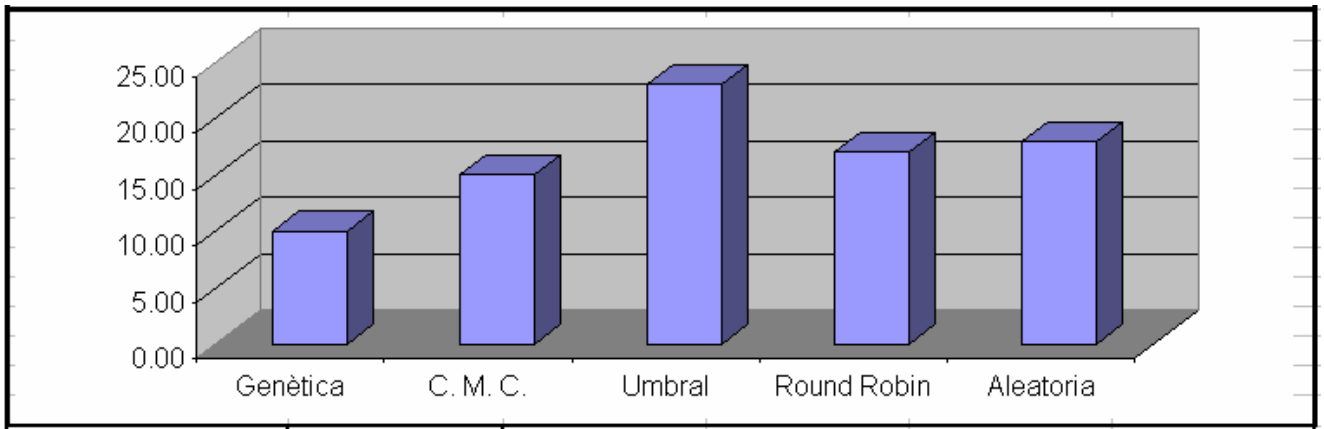


Figura.7.17 Antes y Después de Aplicar un Operador Laplaciano [Lyon 99].

```
public void laplacian3() {
        float k[][] = {
                { 0, -1,  0},
                {-1,  4, -1},
                { 0, -1,  0}
                };
        convolve(k);
  }
```

La Figura 7.17 muestra el mandril antes y después de aplicar un operador Laplaciano, la imagen fue previamente pre-procesada, aplicamos la escala de grises, enseguida un filtro paso bajas, típicamente el filtro consiste en la suma de los pesos de todos los píxeles en la vecindad, dividido entre algún factor de escala. El factor de escala intenta que todos los elementos de los pesos sumen uno o sea que aplicamos el filtro promedio. Después de realizarle el pre-procesamiento a la imagen aplicamos el operador Laplaciano descrito previamente, y posteriormente invertimos la imagen con el propósito de visualizar con mayor claridad los bordes del mandril.

La Figura 7.18 muestra el comportamiento de las estrategias cuando aplicamos el operador Laplace a 800 imágenes heterogéneas en diferentes formatos (GIF y JPEG), como podemos observar en la figura, la estrategia genética, mejora el tiempo de respuesta en cinco minutos en comparación con la estrategia de la Cola Más Corta (CMC) y en un poco mas de 13 minutos si la comparamos con la otra estrategia adaptativa (Umbral). También observamos como la estrategia genética genera tiempos de respuesta bastante considerables en comparación con las estrategias no-adaptaivas (Round_Robin y Aleatoria).



Estrategia	Genètica	C. M. C.	Umbral	Round Robin	Aleatoria
Tiempo (Min.)	10.02	15.02	23.03	17.03	18.03

Figura 7.18 Tiempo de Respuesta (Operador Laplace). Balanceo Dinámico de Carga Heterogéneo en el Procesamiento Distribuido de Imágenes.

7.6.3 Operador Roberts.

Los operadores de gradiente (derivada) en general tienen el efecto de magnificar el ruido subyacente en la imagen. Para compensar ese efecto, tanto el operador de Roberts como el resto de los operadores de extracción de vecindad, utilizados como extractores de bordes, tienen la propiedad añadida de suavizar la imagen, eliminando parte del ruido subyacente y, por lo tanto, minimizan la aparición de bordes falsos debido al efecto de magnificación de ruido por parte de los operadores de derivada [Lyon 1999].

Sobre una ventana *w* de 2*2, asignamos píxeles a una matriz lineal, p:

$$w = \begin{pmatrix} p(0) & p(1) \\ p(2) & p(3) \end{pmatrix} = \begin{bmatrix} f(x)(y) & f(x+1)(y) \\ f(x)(y+1) & f(x+1)(y) \end{bmatrix} \tag{7.33}$$

Definimos la diferencia de operadores sobre las diagonales principal y secundaria como

$$\Delta u = p(0) - p(3), \Delta v = p(1) - p(2) \tag{7.34}$$

El módulo es aproximado, usando:

$$\left|\Delta f(x, y)\right| = \sqrt{\Delta u^2 + \Delta v^2} \tag{7.35}$$

Este es llamado el operador de cruzamiento de Roberts [Pajares 2004]. Desde el punto de vista de la implementación, típicamente comenzamos con una imagen en color. La imagen en color es convertida en una imagen de escala de grises y la detección de bordes es ejecutado usando (7.35). La Figura 7.19 muestra el resultado de aplicar el método de Roberts2, el método *colorToRed* procesa el valor promedio de cada píxel y copiamos este al plano rojo:

Figura.7.19 Antes y Después de Aplicar un Operador Roberts [Lyon 99].

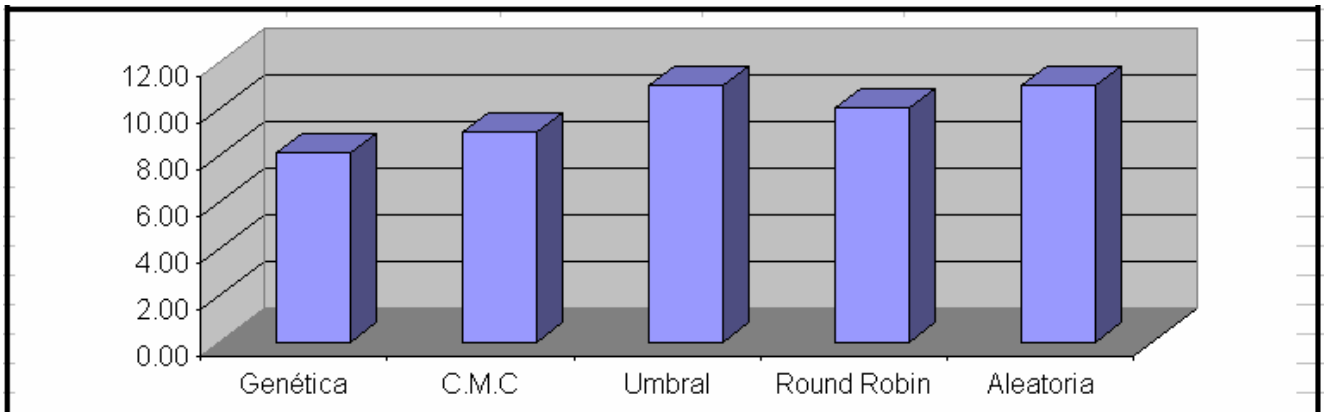
```
public void colorToRed() {
        for (int x=0; x < width; x++)
                for (int y=0; y < height; y++)
                        r[x][y] = (short)
                                ((r[x][y] + g[x][y]  + b[x][y]) / 3);
}
```

El método roberts2 es como sigue:

```
public void roberts2() {
  colorToRed();
  int p[] = new  int[4];
  float delta_u = 0;
  float delta_v = 0;
  short t;
  for (int x=0; x < width-1; x++)
        for (int y=0; y < height-1; y++) {
                p[0] = r[x][y];
```

```
                p[1] = r[x+1][y];
                p[2] = r[x][y+1];
                p[3] = r[x+1][y+1];
                delta_u = p[0] - p[3];
                delta_v = p[1] - p[2];
                t = (short)
                  Math.sqrt(delta_u*delta_u + delta_v*delta_v);
                r[x][y] = t;
                g[x][y] = t;
                b[x][y] = t;
  }
  short2Image();
}
```

La matriz *p* del método *roberts2* es colocada de acuerdo con (7.33). El gradiente es procesado usando (7.34) y es colocado dentro de *t*.



Estrategia	Genética	C.M.C.	Umbral	Round Robin	Aleatoria
Tiempo (Min.)	8.13	9.02	11.01	10.01	11.01

Figura 7.20 Tiempo de Respuesta (Operador de Roberts). Balanceo Dinámico de Carga Heterogéneo en el Procesamiento Distribuido de Imágenes.

La Figura 7.20 muestra el comportamiento de las estrategias cuando aplicamos el operador de Roberts a 800 imágenes heterogéneas en diferentes formatos (GIF y JPEG), como podemos observar en la

figura, la estrategia genética, mejora el tiempo de respuesta en casi cuatro minutos en comparación con la estrategia de la Cola Más Corta (CMC) y en casi 9 minutos si la comparamos con la otra estrategia adaptativa (Umbral). También observamos como la estrategia genética genera tiempos de respuesta bastante considerables en comparación con las estrategias no-adaptaivas (Round_Robin y Aleatoria).

7.7 Conclusiones.

El desarrollo de equipos cada vez más sofisticados en diversos campos de aplicación donde las imágenes constituyen la base hace que el tratamiento de las imágenes sea algo inherente al desarrollo tecnológico. A modo de ejemplo, citemos dos casos de especial relevancia: a) medicina, donde los sofisticados equipos PET (Positron Emisión Tomography), resonancia magnética, rayos X, etc., se verían mermados en su potencialidad de diagnosis de no ser por un tratamiento apropiado de las imágenes digitales que generan; b) observación de la tierra, donde los sensores acoplados en los satélites artificiales son capaces de proporcionar imágenes en las que se pueden observar detalles de hasta 0.6 metros de tamaño; gracias al tratamiento de las imágenes que generan es posible detectar zonas deforestadas, evolución de fenómenos meteorológicos, etc. El objetivo de este trabajo fue doble: por una parte mostramos como se pueden adaptar diferentes métodos del tratamiento de imágenes a nuestra arquitectura y por otra demostramos como nuestra estrategia genera mejores resultados en este tipo de aplicaciones utilizando como métrica de desempeño el tiempo global y el rendimiento del sistema.

Capítulo 8. Análisis Comparativo de las Estrategias de Balanceo de Carga en un Sistema de Verificación Automática de Firmas Manuscritas Off-Line.

En este capítulo se realiza un análisis comparativo de las estrategias de balanceo de carga en un sistema de verificación de firmas *off-line* [Bigus 01], [Hilera 00], [Justino 01], [Lee 96], [Martínez 04], y [Plamondon 00]. El sistema de verificación fue basado sobre una arquitectura que opera en dos fases, la fase de entrenamiento y la fase de verificación. La fase de entrenamiento es formada por los estados de extracción de la firma, preprocesamiento y extracción de características y la generación del modelo. La fase de verificación tiene etapas que son altamente similares a la fase de entrenamiento, en esta se comparten las tres primeras etapas y en la cuarta etapa las características extraídas de la firma son comparadas utilizando un clasificador de redes neuronales artificiales BPNN (Back-Propagation Neural Network) versus las características obtenidas en la fase de entrenamiento. Después de conocer el análisis comparativo de nuestras estrategias en el sistema de verificación, mostramos el trabajo a futuro que pretendemos realizar para lograr el equilibrio de carga en un sistema distribuido de gran escala. Un modelo Fuzzy-Logic en conjunto con el algoritmo distribuido Request for Bids es propuesto para lograr dicho equilibrio de carga [Yen 99], [Yu 04] y [Ferber 99]. Los resultados simulados preliminares son alentadores ya que demuestran como esta propuesta puede ser factible, concluimos el capitulo dando a conocer nuestras conclusiones generales y algunas publicaciones que se realizaron acerca de nuestro trabajo.

8.1 Introducción.

Verificar una firma manuscrita consiste en determinar si, dado un conjunto de ejemplares de la firma de una persona, un ejemplar adicional de dicha firma fue realizado por la misma persona. En este sentido, la verificación de firmas se convierte en un autentificador de la personalidad. Una firma puede ser verificada *on-line* y *off-line*. En el primer caso, una pluma instrumentada o una tableta digitalizadora es utilizada para “capturar” la forma de la firma y la dinámica del movimiento de la mano [Martínez 04], requiriendo por supuesto de la presencia del firmante.

La técnica off-line se refiere a situaciones en que la firma fue realizada previamente sobre papel y es capturada como imagen [Martínez 04], por lo que la riqueza de la información dinámica es perdida y

es básicamente irrecuperable. En ambos métodos, una vez que se cuenta con la firma, se procede a extraer de ella un conjunto de características que deben ser confiables tanto para reconocer firmas genuinas como para rechazar falsificaciones, aún las del tipo bien realizadas (skilled forgeries); el conjunto de características es calculado o extraído de cada una de las firmas de muestra, con lo cual se conforma un conjunto de patrones que a la vez debe servir para el entrenamiento y prueba de un clasificador. El trabajo del clasificador es "aprender" el comportamiento habitual de las características en una firma para posteriormente "comprobar" si dichas características se comportan "igual" en una firma de prueba. En términos de problema de investigación, la verificación de firmas off-line es un problema en el cual los desempeños que se pueden obtener no pueden ser tan altos como en el caso de la verificación on-line, debido a la falta de información dinámica.

8.1.1 Trabajo Previo en Verificación de Firmas Manuscritas *Off-Line*.

La verificación de firmas fuera de línea ha sido un problema de investigación vigente desde hace varias décadas y en diversos países [Plamondon 00], con muchas aplicaciones prácticas potenciales. En [Justino 01], los autores identifican las tres clases de falsificaciones que se describen en la Tabla 8.1.

No.	Nombre	Descripción
1	Falsificación Aleatoria	No se hace intento para reproducir el aspecto de la firma genuina. No se desea parecido entre la firma autentica y la falsificación.
2	Falsificación Simulada	La firma es copiada descuidadamente; no muy detallada o precisa. La firma falsa "tiende" a parecerse a la genuina.
3	Falsificación Hábil	La firma es muy similar a la genuina.

Tabla 8.1 Tipos de Falsificaciones Acorde a Justino [Justino 01].

En la literatura relativa a la verificación *off-line* generalmente se han obtenido buenos desempeños (porcentaje de error de clasificación) ante la falsificación aleatoria, como en [Justino 01] del orden del 0.38% o en Drouhard del 3% [Martínez 04]; sin embargo, ante la falsificación simulada y la falsificación hábil el porcentaje de clasificación errónea crece dramáticamente, y por sólo mencionar

un caso citemos a Fang en [Martínez 04] quien reporta hasta un 33.7%; sin embargo, esta situación no está relacionada con la habilidad del investigador sino con la naturaleza misma del problema de verificación *off-line*, en gran medida por la pérdida de la información dinámica; un factor que agrava el problema de los altos índices de error ante la falsificación hábil es debido a que generalmente debe prepararse un modelo de firmar de un individuo basado solamente en unas pocas muestras, tantas como 8 a 15 [Plamondon 00], lo cual genera incertidumbre probabilística tanto para reconocer firmas genuinas como para rechazar falsificaciones.

Otro de los problemas principales en la verificación *off-line* es el de establecer y extraer de las firmas de muestra un conjunto de características que sea lo suficientemente repetitivo como para permitir alta capacidad de reconocimiento de especimenes genuinos de prueba y lo suficientemente discriminante como para rechazar falsificaciones. En la literatura del tema se pueden encontrar básicamente tres aproximaciones para la generación de características; una consiste en aislar y caracterizar algunos segmentos de la firma (curvatura, smoothness) y otra bien superponer una cuadrícula (gris) a la firma y considerar cada elemento (recuadro) de la cuadrícula como un área a ser caracterizada; un ejemplo clásico de este último lo tenemos en Sabourin 1994 y Sabourin 1997 [Martínez 04]; en ambos casos se busca una representación explícita, a través de vectores descriptivos de los valores de cada característica. En ocasiones se ha permitido traslapamiento de los recuadros que se consideran para representar la firma, como en Murshed 1995, en el cual Murshed y otros usaron cuadrícula de 16x16 píxeles con un traslapamiento del 50% [Martínez 04]. Finalmente se encuentran las aproximaciones basadas en esquemas en los que las características están implícitas en parámetros de alguna clase, como en el caso de Gouvêa 1999 [Martínez 04], quien usó redes neuronales en su versión autoasociativa.

En la última etapa, la del clasificador, se trata de decidir si la firma es genuina o no. Redes neuronales de diversos tipos se han utilizado [Plamondon 00]; y modelos ocultos de Markov [Justino 01] y de otros clasificadores menos sofisticados, como el de los k-vecinos más cercanos y el distancia mínima con medida de Mahalanobis [Martínez 04]. Sin importar el tipo de clasificador, los resultados reportados ante falsificaciones hábiles siguen muy por debajo que los logrados bajo verificación *on-line* [Martínez 04].

8.2 Redes Neuronales.

La red neuronal biológica (cerebro) es la responsable de todos los fenómenos de pensamiento, emoción, aprendizaje y de realizar las funciones sensoriales y motoras. El elemento básico de esta red es la *neurona* que consta básicamente de un *cuerpo celular* más o menos esférico del que sale una rama principal llamada *axón*, y varias ramas más cortas llamadas *dendritas*, el punto de contacto entre un axón de una célula y una dendrita de otra célula es llamado *sinapsis* (ver Figura 8.1).

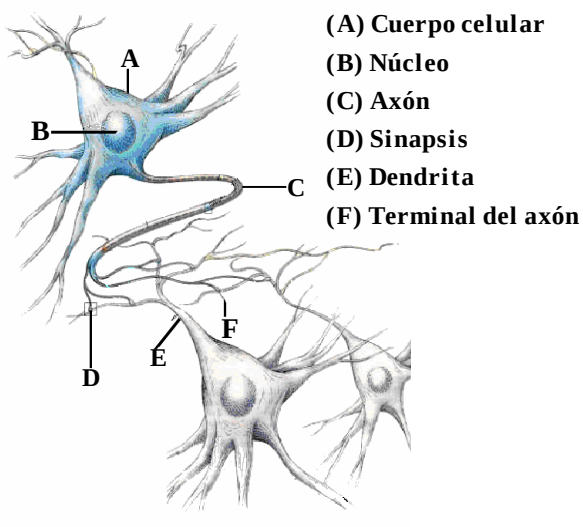


Figura 8.1 Estructura de las Neuronas del Cerebro Humano [Hilera 00].

Una de las características que diferencian las neuronas del resto de las células vivas, es su capacidad para comunicarse. En términos generales, las dendritas y el cuerpo celular reciben las señales de entrada; el cuerpo celular las combina e integra y emite señales de salida. El axón transporta las señales a los terminales axónicos, que se encargan de distribuir la información a un nuevo conjunto de neuronas. Por lo general, una neurona recibe información de miles de otras neuronas y, a su vez, envía información a miles de neuronas más [Hilera 00].

8.2.1 Red Neuronal Artificial (RNA).

Basada fuertemente en la metáfora del cerebro humano, una red neuronal tiene cientos o miles de procesadores (llamados unidades de procesamiento, elementos de procesamiento o neuronas)

conectadas por cientos o miles de pesos adaptativos como se ilustra en la Figura 8.2. Aquí la neurona desempeña un trabajo relativamente pequeño: recibe las entradas de sus neuronas vecinas o de fuentes externas y usa estas para computar una señal de salida, la cual es propagada hacia otras neuronas.

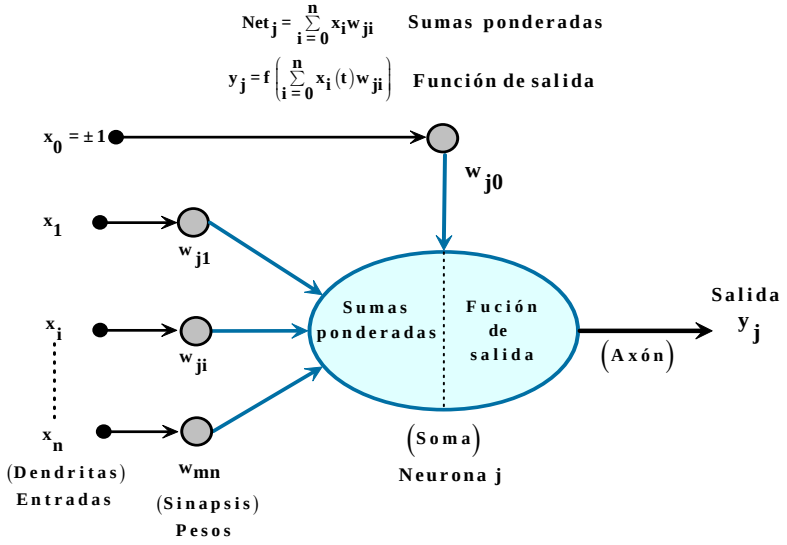


Figura 8.2 Representación Esquemática de una Neurona Artificial [Hilera 00]

Una red neuronal provee una fácil forma de agregar habilidad de aprendizaje a una máquina. Hay muchos diferentes tipos de redes neuronales, algunas se entrenan muy rápidamente y algunas requieren más pasadas sobre los datos antes que aprendan las tareas asignadas.

8.2.2 Tipos de Aprendizaje en la Redes Neuronales Artificiales.

Hay varios paradigmas o enfoques. Estos incluyen aprendizaje supervisado, no supervisado y aprendizaje por reforzamiento. Muchos investigadores y desarrolladores de aplicación combinan dos o más enfoques dentro de un sistema [Bigus 01].

Aprendizaje supervisado. Aquí la red neuronal cuenta con el apoyo externo de un "maestro" que le informa si la salida producida por la red coincide con la salida deseada [Hilera 00]. La red hace una predicción basada sobre la entrada y si la salida difiere de la salida deseada la red es ajustada para producir la salida correcta. Este proceso es repetido una y otra vez hasta que la red aprende a hacer clasificaciones o predicciones exactas.

Aprendizaje no supervisado. Es usado cuando la red necesita reconocer similitudes entre las entradas de los datos. Los datos son presentados a la red y adaptados en particiones grupales. El agrupamiento o proceso de segmentación continúa hasta que la red coloca los mismos datos en los mismos grupos. En este caso no existe un maestro, es decir, la red neuronal no conoce a priori las salidas que debe generar, por lo tanto la red neuronal deberá extraer sin ayuda características de los datos que se le suministran.

Aprendizaje por reforzamiento. Es un tipo de aprendizaje supervisado usado cuando las entradas o salidas de datos no están disponibles. Este puede ser usado en casos donde hay una secuencia de entradas y la salida deseada es solamente conocida después de que la secuencia especificada ocurre. Este proceso identifica la relación entre una serie de valores de entrada y un valor posterior de salida que es llamado asignación de crédito temporal.

8.2.3 Red Neuronal BPNN.

Back-Propagation es la arquitectura más popular de red neuronal para el aprendizaje supervisado. Las características de topología de conexión feed-forward, significa que los datos fluyen a través de la red en una sola dirección, y usa una técnica llamada backward propagation de errores, para ajustar la conectividad de los pesos. Además de tener una capa de entrada y una de salida, una red Back-Propagation puede tener una o mas capas escondidas. Una red Back-Propagation con una simple capa escondida de procesamiento puede aprender a modelar funciones continuas cuando se tienen suficientes unidades en la capa escondida (ver Figura 8.3).

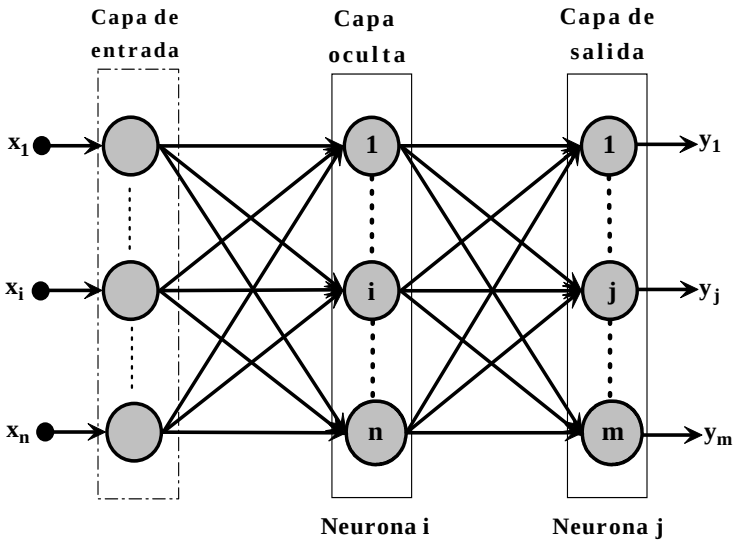


Figura 8.3 Arquitectura de una Red BPNN

Pasos de Entrenamiento.

La Figura 8.3 muestra un diagrama de una red neuronal Back-Propagation e ilustra tres pasos en el proceso de entrenamiento.

Primero. Los datos son presentados en la capa de entrada y fluyen a través de la red hasta alcanzar las unidades de salida. Este es llamado paso hacia delante (forward).

Segundo. La activación o valores de salida representan la salida actual o predicado de salida de la red. La salida deseada es también presentada a la red, por que esta es de aprendizaje supervisado.

Tercero. La diferencia entre la salida deseada y la salida actual es computada, produciendo el error de la red. Este error es entonces pasado de regreso a través de la red para ajustar los pesos.

Cada entrada de la red toma un simple valor numérico, x_i, que es usualmente escalado o normalizado a un valor entre 0.0 y 1.0. Este valor llega a ser la entrada de la unidad de activación. Enseguida necesitamos propagar los datos hacia delante (forward), a través de la red neuronal. Por cada unidad de la capa escondida, computamos la suma de los productos de las unidades de entrada y los pesos conectados a esas unidades de entrada hacia la capa escondida. La suma es el producto (también

llamado punto o producto escalar) para el vector de entrada y los pesos de la capa escondida. Una vez que la suma es computada, agregamos el valor del umbral (threshold) y entonces pasamos esta suma a través de la función de activación no lineal, f, produciendo la unidad de activación y_i. La fórmula para computar la unidad de activación en la capa escondida o de salida dentro de la red es:

$$y_i = f\left(sum_j = \sum x_i w_{ij} + \theta_i\right) \tag{8.1}$$

Donde los rangos de i sobre todas las unidades nos lleva dentro de la unidad j, y la función de activación:

$$f\left(sum_j\right) = \frac{1}{1+e^{-sum_j}} \tag{8.2}$$

Usamos la sigmoide o función logística para f.

La fórmula para calcular los cambios de los pesos.

$$\Delta w_{ij} = \eta \delta_j y_i \tag{8.3}$$

Donde w_{ij} es el peso conectando una unidad i a una unidad j, η es el parámetro de la razón de aprendizaje, δ_j es la señal de error para esa unidad, y y_i es la salida o valor de activación para la unidad i. Para las unidades en la capa de salida, la señal de error es la diferencia entre la salida deseada t_j y la salida actual y_i multiplicada por la derivada de la función de activación logística.

$$\delta_j = \left(t_j - y_j\right) f_j{}'\left(sum_j\right) \tag{8.4}$$

Por cada unidad en la capa escondida, la señal de error es derivada de la función de activación multiplicada por la suma de los productos de los pesos y su correspondiente señal de error. Por lo tanto para la unidad j escondida.

$$\delta_j = f_j'(sum_j)\sum \delta_k w_{jk} \tag{8.5}$$

Donde k es el rango de los índices recibidos para las unidades de salida j´s.

Una modificación común de la regla de los pesos actualizada es el uso del término momento α, para evitar la oscilación de los pesos. Por lo tanto, el cambio de los pesos, llega a ser una combinación del cambio de peso actual, mas una fracción (rangos α de cero a 1) del cambio de peso anterior. Esto complica la implementación por que nosotros ahora tenemos almacenados los cambios de pesos de un paso anterior.

$$\Delta w_{ij}(n+1) = \eta \delta_j y_i + \alpha \Delta w_{ij}(n) \tag{8.6}$$

Cuando el peso cambia es sumado sobre una presentación entera de el conjunto de entrenamiento, la función de minimización del error ejecutada es llamada gradiente descendente. En la práctica, la mayoría de la gente actualiza los pesos de la red después de que cada vector de pesos de entra es presentado.

8.3 Arquitectura del Verificador.

El verificador de nuestro reconocedor automático de firmas manuscritas *off-line* se muestra en la Figura 8.4; se distinguen dos fases: la de entrenamiento (parte superior) y la de verificación (parte inferior). El objetivo de la fase de entrenamiento es generar un modelo matemático por cada firmante, mientras que en la fase de verificación lo es realizar la verificación misma. A continuación se describen los bloques constitutivos de cada etapa.

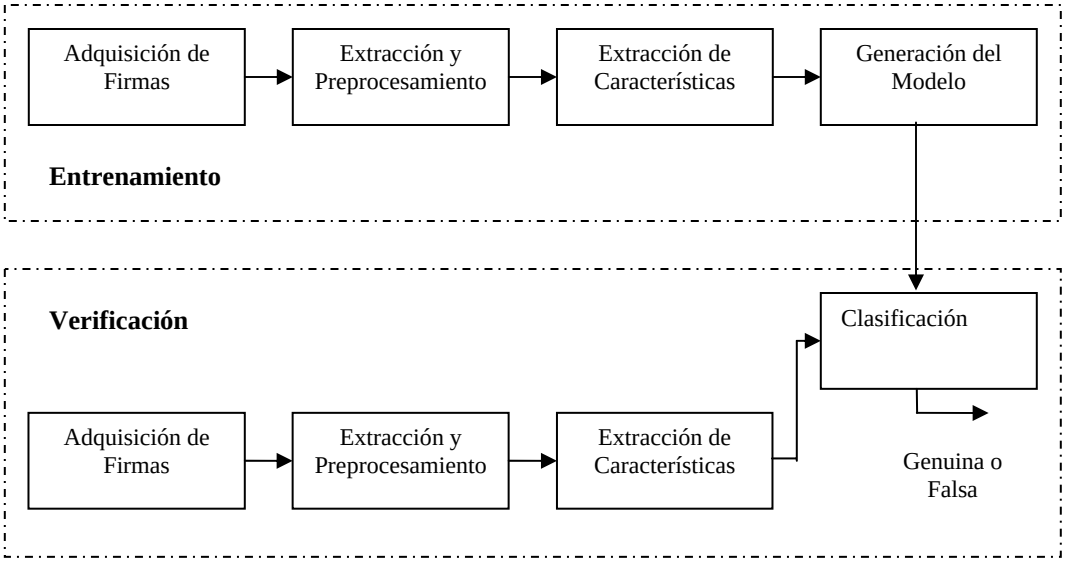


Figura 8.4 Arquitectura del Verificador.

8.3.1 Fase de Entrenamiento.

Los bloques de la fase de entrenamiento se encuentran en la parte superior de la Figura 8.4 y se describen a continuación.

8.3.1.1 Adquisición de las Firmas.

El primer bloque es el de adquisición de imágenes con firmas, y es efectuado con un escáner. Con el escáner se adquiere a una resolución de 300 dpi´s una imagen que contiene hasta 12 firmas. Las firmas son escritas en una hoja blanca de papel tamaño carta; las firmas son escaneadas en escala de grises para hacer el proceso insensible a las variaciones en el color de la tinta de la pluma. Un ejemplo para un firmante ficticio “John Doe” se muestra en la Figura 8.5.

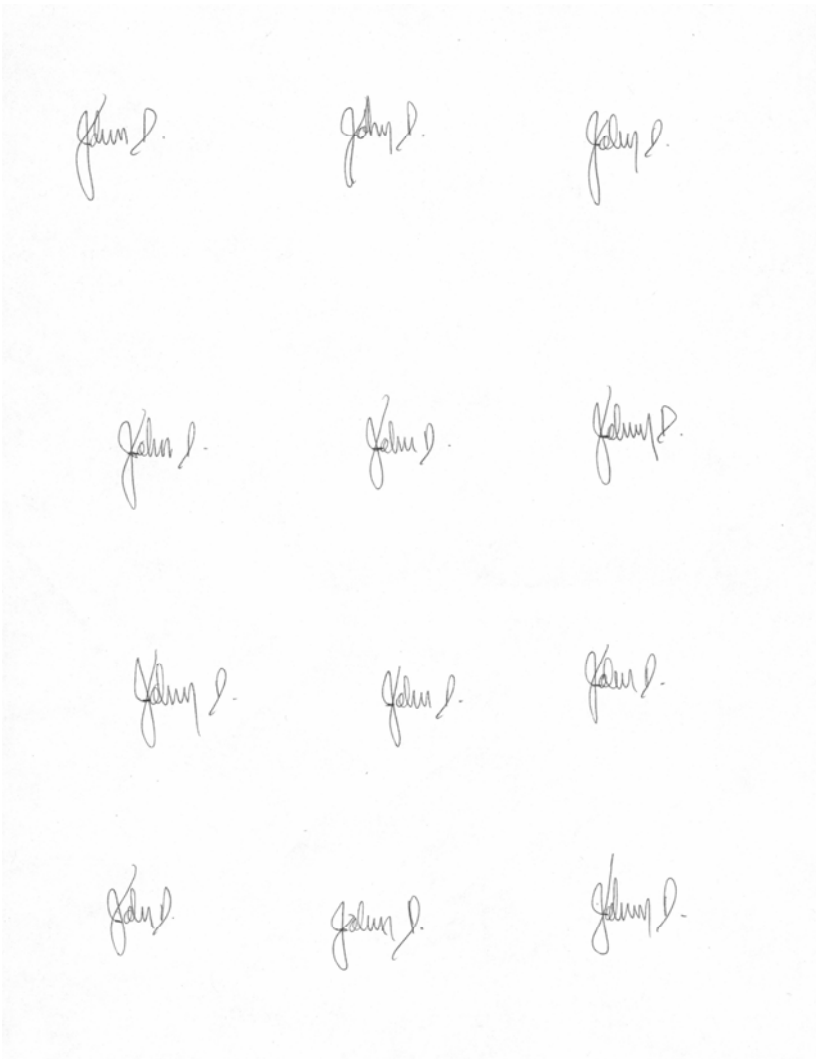


Figura 8.5 Adquisición de una Imagen con 12 Firmas Manuscritas Off-Line a 300 dpi´s.

8.3.1.2 Extracción de las Firmas y Preprocesamiento.

La imagen escaneada contiene 12 firmas igualmente espaciadas, por lo que la extracción de la firma del fondo no es problemática y se efectúa simplemente recortando sub-imágenes de la imagen completa; cada sub-imagen cuenta con una firma. Ver algunos ejemplos de este proceso en la Figura 8.6.

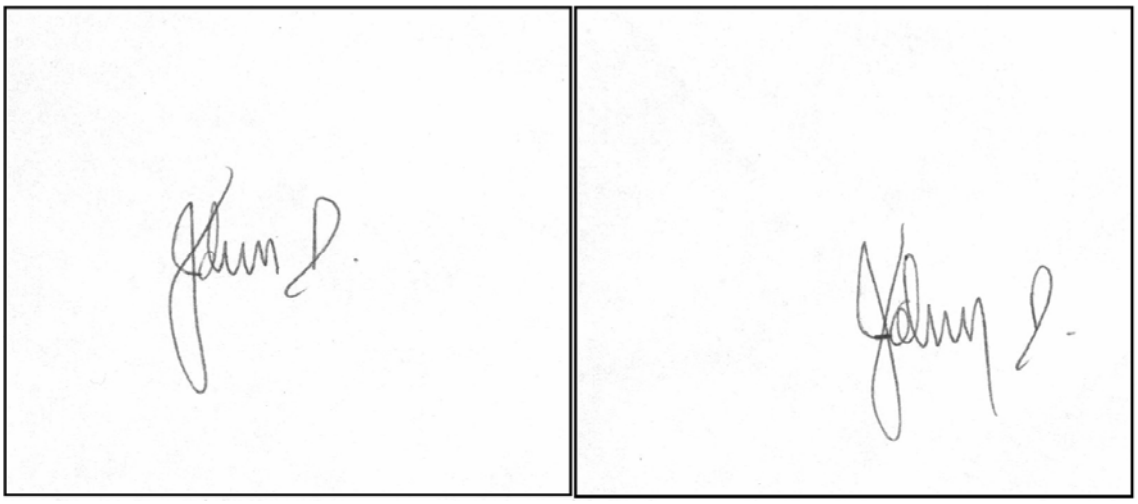


Figura 8.6 Firmas Extraídas del Fondo de la Página.

Binarizar la imagen utilizando un umbral específico por imagen**.** Una mayor explicación sobre la binarización de una imagen se puede observar en la unidad 7, específicamente en el apartado 7.2.4.

8.3.1.3 Extracción de Características de la Firma.

En esta parte de la arquitectura se extraen las características discriminantes. Para extraer las características de la imagen de la firma (primeramente se deben de definir las características y después ejemplificamos como extraerlas).

8.3.1.3.1 Definición de las Características.

Dado que solamente se cuenta con la información estática de la firma manuscrita, el problema de verificar una firma es de mayor complejidad que en la verificación *on-line* si la meta es rechazar falsificaciones hábiles. Nuestra estrategia de verificación está basada en el método de verificación de firmas manuscritas usado por expertos humanos. Los elementos en los que se basa un verificador humano, de acuerdo con Slyter 1995 [Martínez 04], incluyen elementos estáticos y dinámicos. Los elementos estáticos tienen qué ver con la forma o el diseño de la firma. Los elementos dinámicos incluyen la presión absoluta, las variaciones de presión y la velocidad, agrupados en lo que Slyter llama “el ritmo”. El ritmo y la forma se mezclan durante la ejecución de la firma en una forma única

para cada individuo, lo cual denota los hábitos desarrollados al realizar consecutivamente y por mucho tiempo su firma; por lo tanto, cualquier intento de verificar una firma debe considerar el balance entre el ritmo y la forma. En [Plamondon 00], Plamondon establece que existe un cierto consenso entre los investigadores del área en el sentido de que las características para verificación automática deberían reflejar, de una forma u otra, las características que los verificadores expertos utilizan; en este trabajo se ha realizado una adaptación de dichas características al problema de verificación por medios computacionales, como a continuación se explica.

Los componentes del ritmo (rythm) son la velocidad, la presión y las proporciones de los "caracteres" en la firma. La velocidad, según se conoce de diversas publicaciones de verificación de firmas on-line Baron 1989, Plamondon 1988 y Fasquel 2001 [Martínez 04], es proporcional al grado de curvatura de un trazo. Entre más recto sea un trazo, más velozmente fue realizado. Los cambios en el sentido de la escritura, esto es, cambiar de izquierda a derecha o de arriba a abajo y viceversa, indican una interrupción en el flujo del movimiento, lo cual eventualmente implica una velocidad de escritura cero. Si el trazo en el papel es continuo entonces dicho trazo describe una curva y la amplitud de la curva es indicativa de la velocidad "promedio" con que fue realizado el trazo. En general, una curva implica baja velocidad. Nótese que para una firma estática esta es una medida cualitativa. Como consecuencia, la distribución de curvas en la firma es un indicativo de la distribución de la velocidad de firmado.

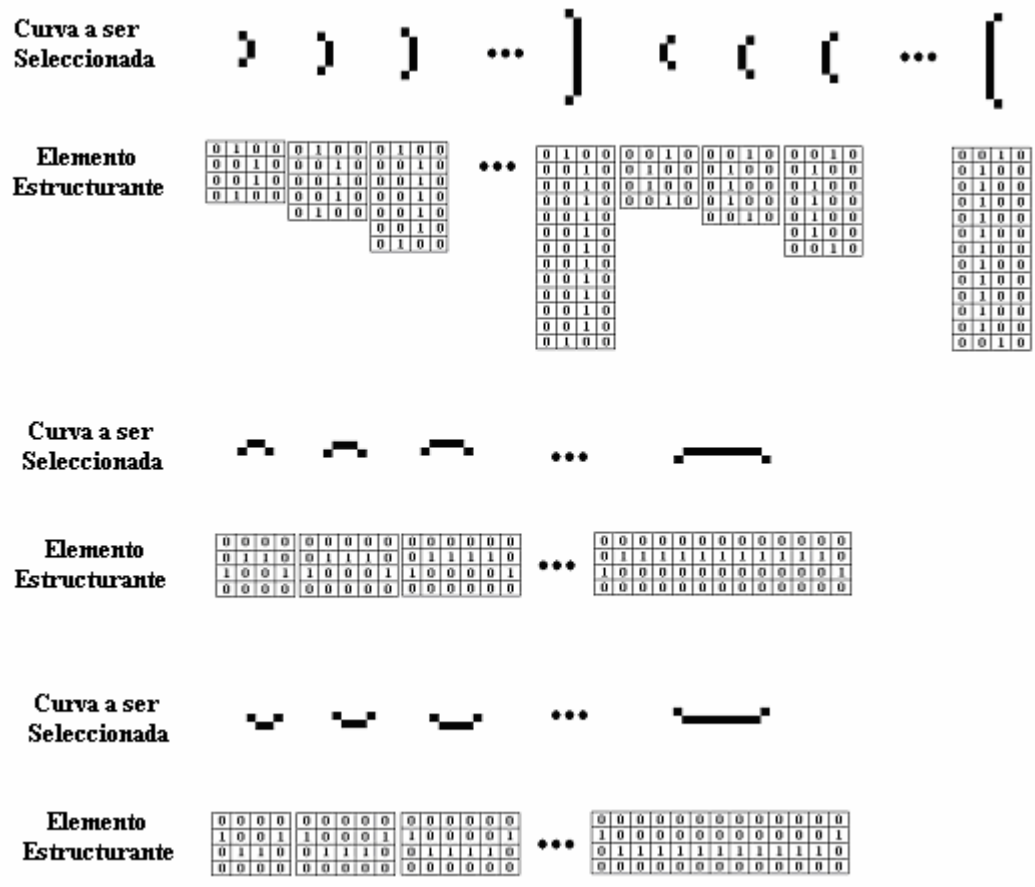


Figura 8.7 Elementos Estructurantes para Detectar Curvas [Martínez 04]

Utilizando morfología matemática y con un conjunto apropiado de elementos estructurantes es posible detectar para una firma cualquiera la posición de los trazos curvos (o equivalentemente, regiones de baja velocidad). La Figura 8.7 muestra un conjunto de elementos estructurantes. La operación morfológica aplicada es la erosión. Al aplicar la erosión a la imagen de la firma con los elementos de la Figura 8.6, se detecta la presencia de trazos curvos y su distribución en la imagen firma. La cantidad de píxeles que quedan "encendidos" en la imagen erosionada pasará a formar parte del *vector de características*. El número total de elementos estructurantes utilizados para las regiones de baja velocidad es de 36.

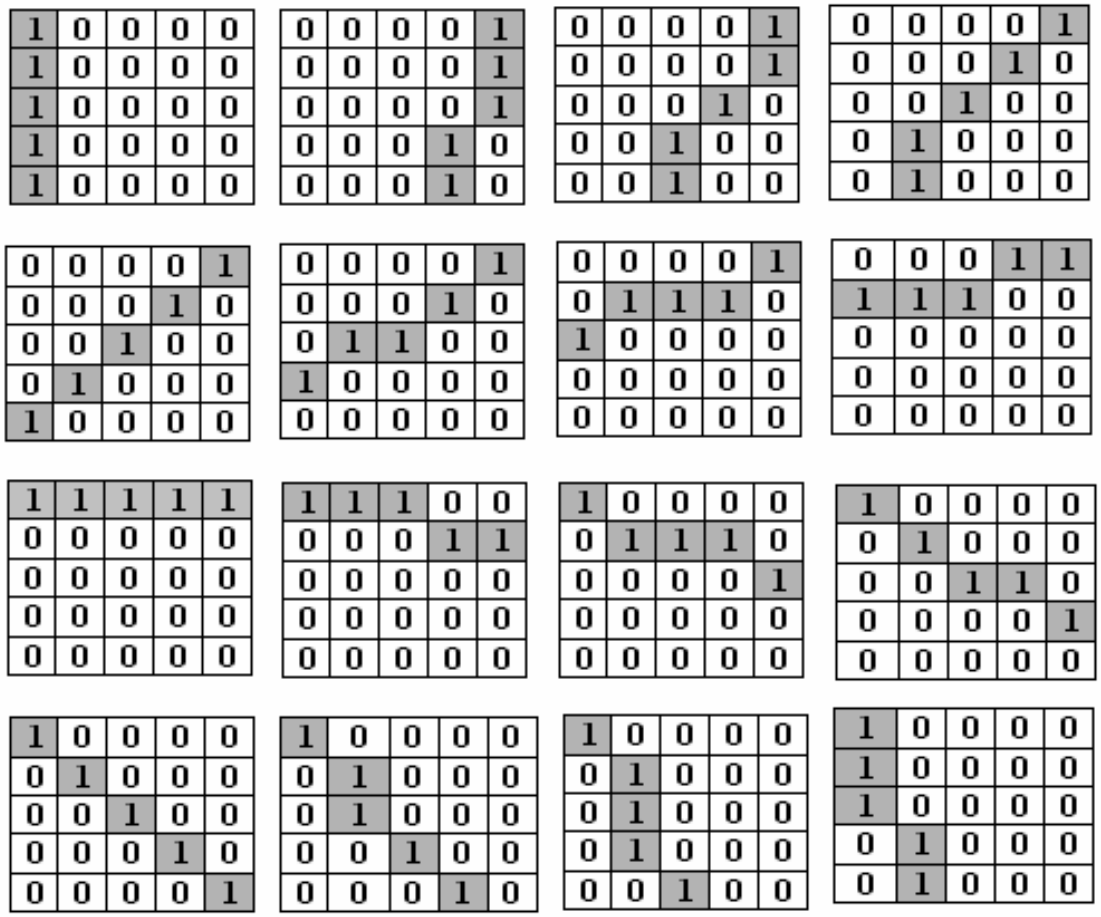


Figura 8.8 Elementos Estructurantes de Alta Velocidad e Inclinación[Martínez 04]

La inclinación general de la firma, la estructura global y las regiones de alta velocidad se encuentran relacionadas con los trazos rectos; los trazos rectos pueden existir en diferentes inclinaciones. En [Lee 96], los autores propusieron un conjunto de elementos estructurantes para detectar trazos. En este trabajo, el conjunto de dichos elementos es retomado y ampliado, y los elementos estructurantes respectivos se muestran en la Figura 8.8.

Las componentes de la forma o diseño son las micro-formas, la forma de los caracteres y las formas conectivas (enlaces entre una parte de la firma y otra). Algorítmicamente hablando, estas características son difíciles de aislar en una firma, y además de un ejemplar al siguiente de una misma firma no siempre aparecen los mismos trazos, o en ocasiones se generan durante el firmado trazos “de ruido”, que no tienen qué ver con el estilo en general adoptado por el firmante. En este trabajo, dado que los elementos estructurantes de las Figuras 8.7 y 8.8 no solamente están orientados a detectar

velocidades, sino que por su naturaleza también describen la forma de la firma, consideraremos los elementos de forma como implícitos en los del ritmo.

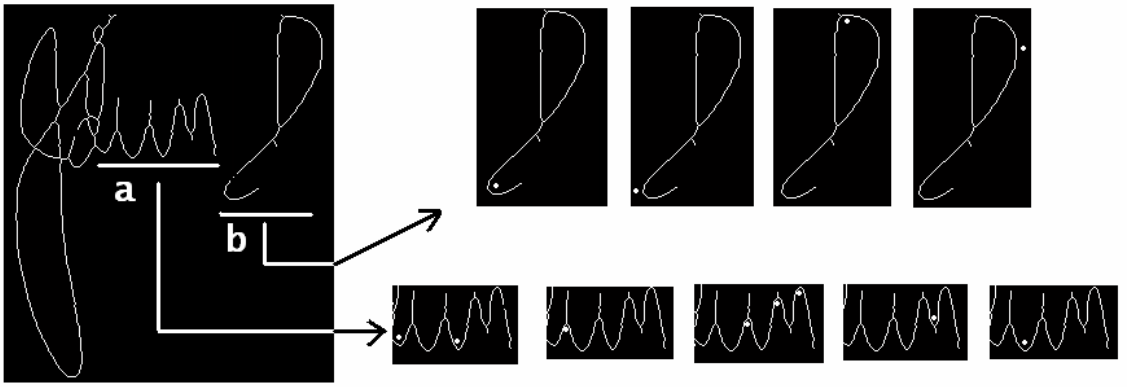


Figura 8.9 Detección de Áreas de Baja Velocidad.

8.3.1.3.2 Ejemplo de Extracción de las Características.

La Figura 8.9 muestra la detección de puntos de baja velocidad en la firma de "John Doe". Nótese en primer lugar los segmentos de firma subrayados y etiquetados como *a* y *b*. El segmento *a* representa típicas ondulaciones presentes en las firmas; el punto indica la posición en que los cambios de sentido en la escritura fueron detectados y representan puntos de muy baja velocidad. En el caso del segmento *b*, el punto indica los cuatro puntos de baja velocidad en el trazo "*D"*; sin embargo, es instructivo notar que, en un orden de izquierda a derecha, los primeros tres puntos de muy baja velocidad (cambio abrupto en el cambio de dirección), mientras que el cuarto es de velocidad moderadamente baja, reflejo de una curvatura mayor.

Una gráfica del vector característico para la firma de "John Doe" se muestra en el lado izquierdo de la Figura 8.10. Cada valor en dicho vector representa el número de píxeles "encendidos" (que valen 1) en la matriz que representa a la imagen erosionada; dicho número es igual al número de ocurrencias del elemento estructural en la imagen. En el lado derecho de la Figura 8.9 se muestra la gráfica de 10 vectores característicos correspondientes a la misma cantidad de firmas de "John Doe". Nótese la repetibilidad en los valores entre los vectores.

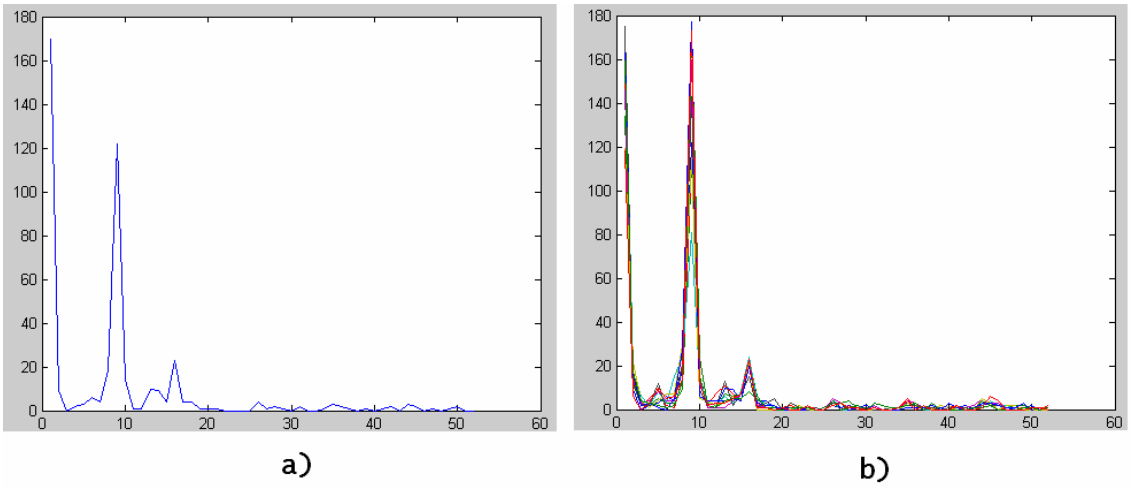


Figura 8.10 Vectores Característicos de 10 Firmas de "John Doe".

8.3.1.4 Generación del Modelo.

En esta sub-sección se explica la generación del modelo de la firma y se incluye el diseño del clasificador.

8.3.1.4.1 Modelo de la Firma y Clasificador.

Después de que la firma ha sido binarizada, aplicamos los siguientes pasos para generar el modelo de la firma y clasificador.

1. Filtrado Morfológico.
2. Generación de Patrones de Entrenamiento.
3. Arquitectura de la Red Neuronal Back-Propagation (Clasificador).
4. Entrenamiento del Clasificador.

1. Filtrado Morfológico.

La idea básica del procesamiento de imágenes morfológicas es que el elemento estructural sea usado para examinar un conjunto de imágenes. El centro del elemento estructural es movido de un píxel al próximo en la imagen. Un conjunto de operaciones es usado entre el elemento estructural y la parte de

la imagen que es traslapada con este elemento. El conjunto de operaciones produce información estructural acerca de la imagen. Históricamente, el procesamiento de imágenes morfológicas es con imágenes binarias y procesamiento de imágenes en escala de grises. En este caso, solo trabajaremos el caso binario y la operación morfológica utilizada es la erosión. El fundamento matemático del filtrado morfológico y una explicación mas detallada de la erosión pueden observarse en el anexo B de este escrito.

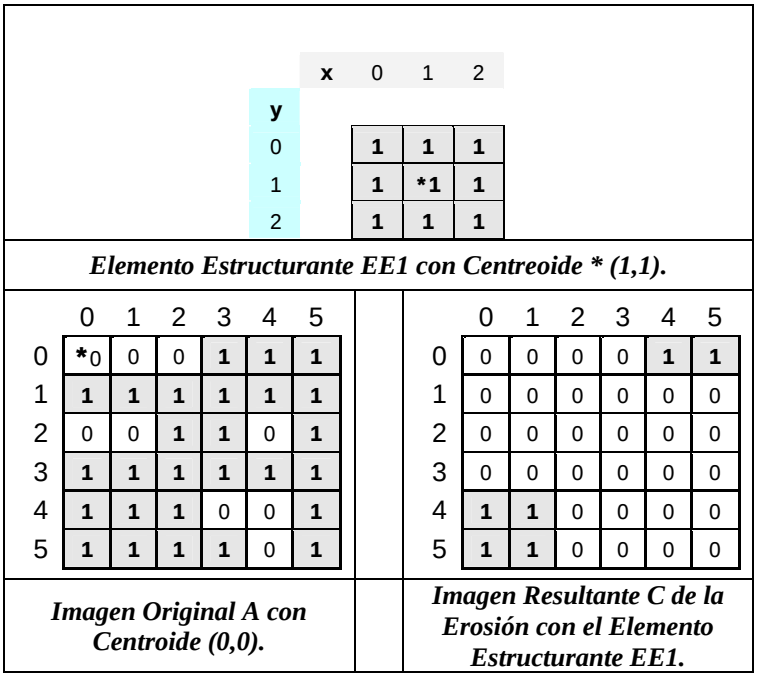


Figura 8.11 Erosión de una Imagen A con EE1 como Elemento Estructurante.

La Figura 8.11 muestra claramente como queda la imagen (imagen C) después de aplicarle la operación morfológica de la erosión con el elemento estructurante EE1. Las coordenadas siguientes denotan los bits encendidos (unos) de cada una de las imágenes:

$$EE1 = \left\{ \begin{matrix} (0,0)\ (1,0)\ (2,0) \\ (0,1)\ (1,1)\ (2,1) \\ (0,2)\ (1,2)\ (2,2) \end{matrix} \right\}$$

$$A = \left\{ \begin{matrix} (3,0)\ (4,0)\ (5,0) \\ (0,1)\ (1,1)\ (2,1)\ (3,1)\ (4,1)\ (5,1) \\ (2,2)\ (3,2)\ (5,2) \\ (0,3)\ (1,3)\ (2,3)\ (3,3)\ (4,3)\ (5,3) \\ (0,4)\ (1,4)\ (2,4)\ (5,4) \\ (0,5)\ (1,5)\ (2,5)\ (3,5)\ (5,5) \end{matrix} \right\}$$

$$EE1 \otimes A = \left\{ \begin{matrix} (4,0)\ (5,0) \\ (0,4)\ (1,4) \\ (0,5)\ (1,5) \end{matrix} \right\}$$

2. Generación de Patrones de Entrenamiento.

En la Tabla 8.2 se observan los patrones de entrada que le serán suministrados a la red neuronal, nótese que cada fila de la tabla es considerado como un patrón (o ejemplo de entrenamiento), por lo tanto de acuerdo a la Tabla 8.2 podemos generar un total 110 patrones de entrenamiento por cada firmante que la red neuronal tendrá que aprender para identificar a un firmante específico. En general la Tabla se encuentra dividida en las siguientes secciones:

Patrones Reales: Corresponden de la fila 1 a la 10 y son formados por la erosión de cada una de las 10 firmas de un solo firmante (nuestro firmante ficticio "John Doe"). Nótese que cada columna de esta sección corresponde a un valor que representa el número del elemento estructurante con el cual se erosiono dicha firma, por lo tanto el número entero que aparece en la columna es la sumatoria de los bits encendidos (unos) que quedaron después de erosionar la firma con el elemento estructurante en cuestión, cada renglón es un vector de características.

Patrones Sintéticos Positivos: Corresponden de la fila 11 a la 60. Para obtener una columna de estos números se generan números aleatorios en un rango de $\frac{1}{10}\sum_{k=1}^{10} x_k \pm \sigma$ en donde x_k es el valor de cada celda EE de la Tabla 4.2 a lo largo de las firmas $F_1, F_2, F_3, ..., F_{10}$.

Patrones Sintéticos Negativos: Corresponden de la fila 61 a la 110. Para obtener una columna de estos números se generan números aleatorios en un rango de 1.0 y 300.0.

Parte del Patrón	Numero de Firma	Elemento Estructurante (ocurrencias en la firma)						
		EE1	EE2	EE3	EE4	EE5	---------	EE54
Real	F1	1153	829	568	448	405	---------	393
	F2	1077	696	447	369	331	---------	227
	F3	1221	817	535	427	368	---------	393
	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-
	F10	1238	856	516	386	317	---------	276
Sintéticos Positivos	F11	1117	745	549	431	270	---------	340
	F12	1135	879	541	311	359	---------	352
	F13	1083	723	450	397	272	---------	270
	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-
	F50	1176	858	460	392	314	---------	339
Sintéticos Negativos	F51	247	252	156	213	247	---------	178
	F52	73	295	221	20	63	---------	274
	F53	80	114	10	226	70	---------	6
	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-
	F110	137	201	47	32	164	---------	226

Tabla 8.2 Ejemplos de Entrenamiento para la Red Neuronal.

3. Arquitectura de la Red Neuronal Back-Propagation (Clasificador).

La arquitectura de la red neuronal de la Figura 8.12 esta compuesta por una capa de entrada con 54 neuronas, una sola capa oculta con 18 neuronas y una capa de salida con 8 neuronas con una función tipo sigmoidal. Como puede observarse en la red neuronal en su capa de entrada, cada neurona recibe un número que corresponde a cada uno de los elementos estructurantes, generando un total de 54 elementos estructurantes que son el mismo número de neuronas en la capa de entrada de la red neuronal, para este caso en particular los números que se le suministran a la capa de entrada corresponden al patrón F1 de la Tabla 8.2. La razón por la cual existen 8 neuronas en la capa de salida es por que ante un patrón de (ejemplo de entrenamiento) estímulo suministrado a la red, esta deberá aproximar su salida deseada a 8 números binarios que representan el identificador del firmante.

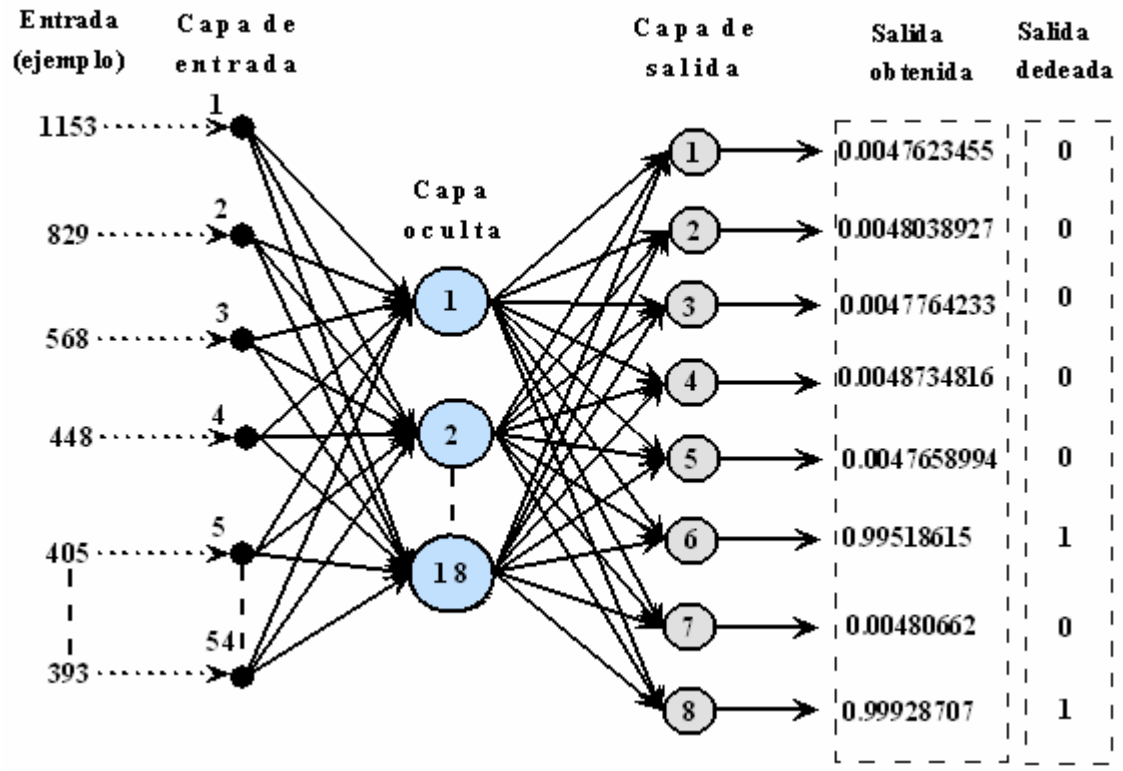


Figura 8.12 Arquitectura de la Red Neuronal Recibiendo una Entrada (Firma) y una Salida Deseada (Vector Binario que Corresponde al Identificador del Firmante).

4. Entrenamiento del Clasificador.

Después de que la arquitectura del clasificador ha sido diseñada el siguiente paso es entrenar el clasificador de tal forma que sirva como herramienta de reconocimiento en la fase siguiente del verificador. Los valores de inicialización de la matriz de pesos se generan aleatoriamente; después de varias pruebas, los parámetros que determinan el entrenamiento se fijaron en los valores con los cuales se alcanzó el mejor rendimiento de la red. El error cuadrático medio se considero como 0.003, el número máximo de iteraciones en caso de no alcanzar el error cuadrático medio fue determinado en 1000 iteraciones, la tasa de aprendizaje se estableció en un valor de 1E-20 para avanzar por la superficie del error con incrementos pequeños de los pesos.

8.3.2 Fase de Verificación.

De la Figura 8.4, parte inferior, es evidente que las fases de entrenamiento y verificación son altamente similares, compartiendo las etapas de adquisición de las firmas, extracción de las firmas y preprocesamiento, la extracción de características de las firmas y algunos pasos de la generación del modelo.

Cuando un conjunto de firmas bajo prueba es presentado al verificador, los siguientes eventos tienen lugar:

1.- Las primeras tres etapas de verificación se ejecutan, las cuales generan las características que dan origen a los patrones de entrenamiento en cada una de las firmas,
2.- Después que los patrones de entrenamiento son generados, el clasificador ya no es entrenado solamente verifica la firma declarando la firma como verdadera, o falsa en caso contrario.

Parte del Patrón	Numero de Firma	Elemento Estructurante (ocurrencias en la firma)							Salida de la Red Neuronal
		EE1		EE3	EE4	EE5	---------	EE54	
Real "Firmante X"	F1	1153	4.6868	568	448	405	---------	393	4.9951
	F2	1077	6.6339	447	369	331	---------	227	4.9992
	F3	1221	4.6552	535	427	368	---------	393	4.9953
	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-
	F10	1238	856	516	386	317	---------	276	4.9993
Sintéticos Negativos "Firmante Y"	F11	98	12	6	9	0	---------	34	1.9233
	F12	152	22	5	12	13	---------	34	2.9956

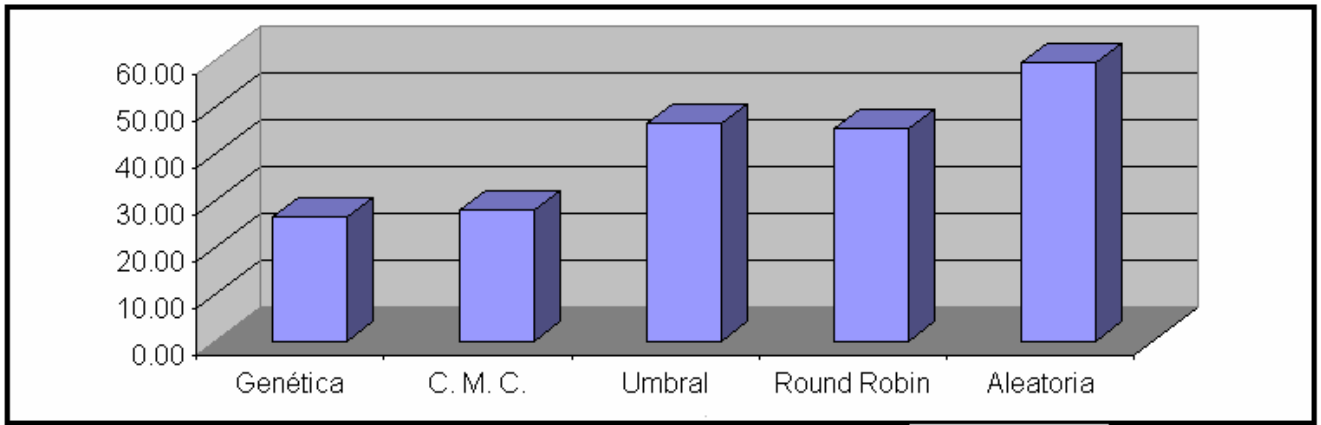
Tabla 8.3 Salidas Obtenidas por la Red Durante la Verificación.

En Tabla 8.3 se pueden observar los resultados obtenidos en la etapa de verificación al suministrarle a la red 10 [1…10] firmas de un firmante "X" y 2 firmas [11…12] de un firmante "Y". En la Tabla 8.3 podemos observar claramente que las 10 primera firmas generan resultados muy similares y los resultados de las últimas 2 firmas son totalmente diferentes, esto se debe a que la red neuronal fue

entrenada para verificar las firmas del firmante “X”.

8.4 Análisis Comparativo de las Estrategias de Balanceo de Carga en el Sistema de Verificación Automática de Firmas Manuscritas Off-Line.

El sistema de verificación de firmas off-line fue implementado en nuestra arquitectura de balanceo de carga y fue evaluado con las diferentes estrategias de balanceo de carga descritas previamente. La Figura 8.13 muestra el comportamiento de las estrategias cuando evaluamos 2400 firmas incluidas en 200 imágenes con formato (BMP), como podemos observar en la Figura la estrategia Genética, mejora el tiempo de respuesta en 1.5 minutos en comparación con la estrategia de la C.M.C. y en 20.01 minutos si la comparamos con la otra estrategia adaptativa (Umbral). También observamos como la estrategia genética genera tiempos de respuesta bastante considerables en comparación con las estrategias no-adaptaivas (Round Robin y Aleatoria), si la comparamos con la estrategia Round_Robin la estrategia Genética mejoró el tiempo de respuesta en 18.97 minutos y en comparación con la estrategia Aleatoria, la estrategia Genética mejoró el tiempo de respuesta en 32.92 minutos.



Estrategia	Genética	C.M.C.	Umbral	Round Robin	Aleatoria
Tiempo (Min.)	26.44	27.94	46.45	45.41	59.36

Figura 8.13 Tiempo de Respuesta. Balanceo Dinámico de Carga en un Sistema de Verificación de Firmas Manuscritas Off-line.

8.5 Conclusiones.

Se presentó un esquema de verificación de firmas manuscritas fuera de línea, que se basó en el fundamento conceptual utilizado por los verificadores humanos expertos para confeccionar un conjunto de elementos estructurantes. Al erosionar una imagen firma binarizada con dichos elementos estructurantes se detectaron diversos patrones de entrenamiento que sirven como entradas a una red neuronal Back-Propagation, que fue entrenada posteriormente con estos patrones y algunos otros patrones complementarios que nosotros decidimos llamar patrones reales, sintéticos y negativos, con esto concluimos la primera fase del sistema a la que nosotros llamamos fase de entrenamiento. Es evidente que las fases de entrenamiento y verificación son altamente similares, compartiendo las primeras etapas y algunos pasos de la generación del modelo. Cuando un conjunto de firmas bajo prueba es presentado al verificador, el clasificador ya no es entrenado solamente verifica la firma declarando la firma como genuina, o falsa en caso contrario.

La verificación de firmas fuera de línea fue implementada en nuestra arquitectura de balanceo de carga y fue evaluado con nuestras diferentes estrategias de balanceo de carga descritas previamente. En la arquitectura nosotros observamos claramente como la estrategia genética obtuvo mejores tiempos de respuesta en comparación con las otras estrategias, confirmando una vez más, que cuando la granularidad y el tiempo de procesamiento se incrementan nuestra estrategia siempre será la mejor opción.

El sistema desarrollado que se implemento en nuestra arquitectura de balanceo de carga reconoce firmas genuinas sin ser todavía capaz de rechazar falsificaciones, por lo tanto, este, podría clasificarse como reconocedor más que como verificador, sin embargo su funcionalidad puede extenderse a la de un verificador si en el conjunto de entrenamiento de la red neuronal se incluye en el conjunto de ejemplos negativos algunas características de firmas genuinas distorsionadas a modo de falsificaciones con la cual la red neuronal aprenderá a distorsionar falsificaciones hábiles.

Conclusiones Generales.

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) [OMG 01]. 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 [Ossama 01a,b], VisiBroker [Lindermeier 00], MICO [Puder 02], 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 [OMG 01b] con el propósito de establecer un estándar de balanceo de carga y monitoreo en aplicaciones basadas en CORBA. El artículo de IONA Technologies [IONA 02] 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 por que ella considera la información de estado actual para hacer decisiones de balanceo de carga.

La idea principal en este trabajo fue la de mejorar LL aplicando una estrategia Genética, creemos que nuestro trabajo es original ya que no hay en la literatura otros trabajos que traten de combinar este enfoque para mejorar el balanceo de carga dinámico bajo CORBA. Nuestra estrategia fue implementada y evaluada en una arquitectura de balanceo de carga construida por nosotros y mostramos que bajo ciertas condiciones del sistema, nuestra estrategia obtuvo un mejor desempeño.

Considerando el tiempo global y el rendimiento del sistema de las gráficas mostradas en el capítulo 6, 7 y 8, podemos concluir que:

En el balanceo dinámico de carga homogéneo:

- Nuestra estrategia es una buena opción si la granularidad(peso) de los requerimientos es delgada y la demanda de los clientes es baja.

- Nuestra estrategia es una excelente opción si la granularidad de los requerimientos es gruesa y la demanda de los clientes es baja.

- Nuestra estrategia es la mejor opción si la granularidad es gruesa y la demanda de los clientes es alta.

En el balanceo dinámico de carga heterogéneo:

- Nuestra estrategia es una excelente opción si la granularidad heterogénea es delgada y la demanda de los clientes es baja.

- Nuestra estrategia es la mejor opción si la granularidad heterogénea es gruesa y la demanda de los clientes es baja.

- Nuestra estrategia es la mejor opción si la granularidad heterogénea es gruesa y la demanda de los clientes es alta.

En el Procesamiento Distribuido de Imágenes.

- Nuestra estrategia es una excelente opción, cuando la granularidad es gruesa y la demanda del cliente es baja.

- Nuestra estrategia es la mejor opción, cuando la granularidad es gruesa y la demanda del cliente es alta.

En el Sistema de Verificación de Firmas Off-line.

- Nuestra estrategia es una excelente opción, cuando la granularidad es gruesa y la demanda del cliente es baja.

- Nuestra estrategia es la mejor opción, cuando la granularidad es gruesa y la demanda del cliente es alta.

Anexo A. Algoritmos Genéticos.

Este anexo tiene como objetivo ilustrar cómo funcionan los Algoritmos Genéticos, describe el funcionamiento del algoritmo genético dando un ejemplo sencillo para diseñar una simple lata. Muestra la representación de cromosomas, función objetivo y los operadores de selección, cruzamiento y mutación en base al problema planteado, tal como lo describe D. E. Goldberg [Golberg 89, 91], Randy L. Haupt and Sue E. Haupt [Haupt 98] y Deb Kalyanmoy [Kalyanmoy 2001].

A.1 Introducción.

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 su perspectiva global [Goldberg 89].

El concepto de AG fue primero concebido por John Holland para la Universidad de Michigan. Después de esto, él y sus estudiantes han contribuido mucho al desarrollo de este campo. La mayoría del trabajo de búsqueda inicial pudo ser obtenido en varios Proceedings de Conferencias Internacionales. Sin embargo ahora existen varios libros de texto sobre AGs [Goldberg 89, Gen 97, Holland 75, Michalewicz 92, Vose 99]. Una descripción más comprensiva de los GA, junto con otros algoritmos evolutivos, puede ser obtenida en el compilado llamado “Handbook on Evolutionary Computation [Back 97]”. Tres Journals (“Evolutionary Computation Journal Published by MIT Press”, “Transactions on Evolutionary Computation Published by IEEE” y “Genetic Programming and Evolvable Machines Published by Kluwer Academic Publishers”), están ahora dedicados a promover la investigación en el campo. Además, la mayoría de las aplicaciones de los AGs pueden también ser obtenidas en varios Journals de dominios específicos.

A.2 Definición de Algoritmo Genético.

Koza define un Algoritmo Genético partiendo de los elementos básicos que deben tener los AGs [Koza 92]:

"El Algoritmo Genético, es un algoritmo matemático altamente paralelo que transforma un conjunto (población) de objetos matemáticos individuales (típicamente cadenas de caracteres de longitud fija que se ajustan al modelo de las cadenas de cromosomas), cada uno de los cuales se asocia con una aptitud, en una población nueva (es decir, la siguiente generación) usando operaciones modeladas de acuerdo al principio Darwiniano de reproducción y sobrevivencia del más apto y tras haberse presentado de forma natural una serie de operaciones genéticas (notablemente la recombinación sexual)."

A.3 Funcionamiento de un AG.

El siguiente modelo nos muestra los pasos básicos que haya que evaluar para el desarrollo de alguna aplicación en donde intervienen los AGs.

1. [Iniciar] Generar una población aleatoria de n cromosomas (mejores soluciones del problema)
2. [Fitness - Elitismo] Evaluar el fitness f(x) de cada cromosoma x en la población y el mejor es agregado a la nueva población
3. [Nueva población] Crear una nueva población repitiendo los siguientes pasos hasta que la población este completa.

3.1 [Selección] Selecciona dos cromosomas padres de la población de acuerdo a su fitness (el mejor fitness, tiene mas oportunidad de ser seleccionado)

3.2 [Cruzamiento] Con una probabilidad de cruzamiento los padres forman unos hijos. Si el cruzamiento no es llevado a cabo, los hijos son iguales a los padres.

3.3 [Mutación] Con una probabilidad de mutación mutar los hijos (en una posición del nuevo cromosoma).

3.4 [Aceptación] Colocar los hijos en la nueva población
4. [Remplazar] Usar la nueva población para una corrida futura del algoritmo
5. [Probar] Si el fin de la condición es satisfecha, detener, y, regresa la mejor solución en la actual población.
6. [Fin de ciclo] Ir al paso 2.

A.4 Algoritmos Genéticos Binarios.

Como su nombre lo dice, los AGs toman sus principios de trabajo de la naturaleza genética [Haupt98]. La Figura A.1 muestra la analogía entre la evolución biológica y un AG binario. Ambos comienzan con una población inicial aleatoria. En la parte izquierda cada renglón de números binarios representa características de uno de los perros en la población. Si estamos intentando crear un perro laudest bark, entonces solamente algunos de los mejores perros loudest bark son seleccionados (en este caso, cuatro mejores).

Características asociadas con laudest bark son codificadas en una secuencia binaria asociada con los perros. De esta nueva población dos son seleccionados aleatoriamente para crear dos nuevos cachorros. Los cachorros tienen altas probabilidades de ser laudest bark por que sus padres tienen genes que los hacen loudest bark. Las nuevas secuencias binarias de los cachorros contienen porciones de las secuencias binarias de ambos padres. Estos nuevos cachorros remplazan los dos perros que no son lo suficientemente loudest bark.

A.5 Trabajando los Principios de los AGs Binarios.

Los AGs son procedimientos de búsqueda y optimización que son motivados por los principios de genética natural y selección natural. Algunas ideas fundamentales de genéticos son prestadas y usadas artificialmente para construir algoritmos de búsqueda que son robustos y requieren mínima información del problema. El principio de trabajo de los AGs es muy diferente de la mayoría de las técnicas de optimización. Describimos el trabajo de un algoritmo genético ilustrando el simple problema de diseño de una lata. Una lata cilíndrica es considerada para tener solamente dos parámetros el diámetro “d” y la altura “h”. Consideramos que la lata necesita tener un volumen de al menos 300ml y el objetivo de diseño es minimizar el costo para el material de la lata. Con esta restricción y el objetivo, primero escribimos el correspondiente problema de programación no lineal.

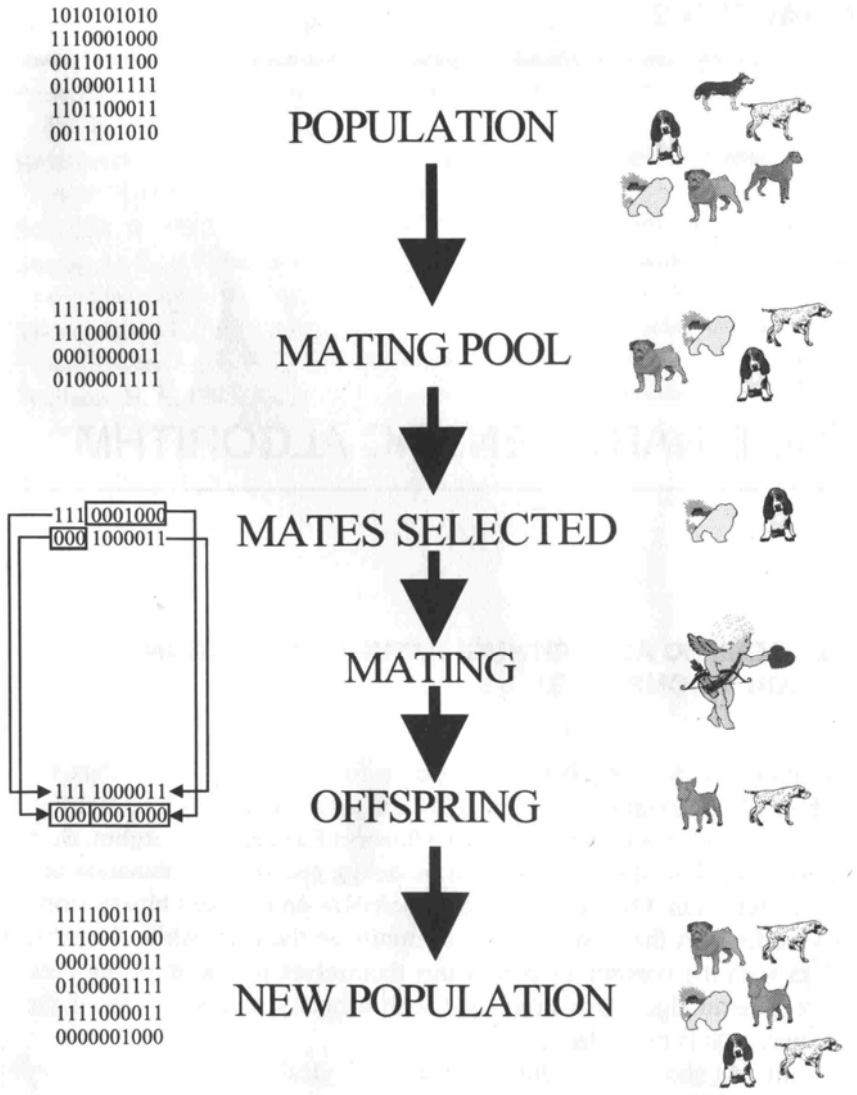


Figura A.1 Analogía Entre un Algoritmo Genético Binario y La Genética Biológica [Haupt 98].

Minimizar $\quad f(d,h) = c\left(\frac{\pi d^2}{2} + \pi dh\right),$

Sujeto a $g_1(d,h) \Leftrightarrow \frac{\pi d^2 h}{4} \geq 300,$

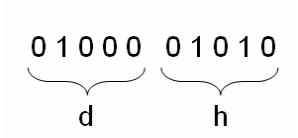
Variables atadas

$$d_{\min} \leq d \leq d_{\max},$$
$$h_{\min} \leq h \leq h_{\max}. \qquad \text{(A.1)}$$

El parámetro c es el costo del material de la lata por centímetro "cm" cuadrado, y las variables de decisión d y h son admitidas para variar en $[d_{\min}, d_{\max}]$ y $[h_{\min}, h_{\max}]$ cm respectivamente.

A.5.1 Representando la Solución.

La forma de usar AGs para obtener las variables de decisión óptima d y h que satisfacen la restricción g_1 y minimizan f, primero necesitamos representarlas en cadenas (strings) binarias. Asumimos que debemos usar cinco bits para codificar cada una de las dos variables de decisión, por consiguiente hacemos la longitud de la cadena igual a 10. La siguiente cadena representa una lata de diámetro 8 cm y altura 10 cm:



Estas cadenas y sus correspondientes variables de decisión son mostradas en la Figura A.2. En esta representación, los límites bajos y altos de ambas variables de decisión son consideradas en cero y 31, respectivamente. Con cinco bits para representar una variable de decisión, hay exactamente 2^5 o 32 diferentes soluciones.

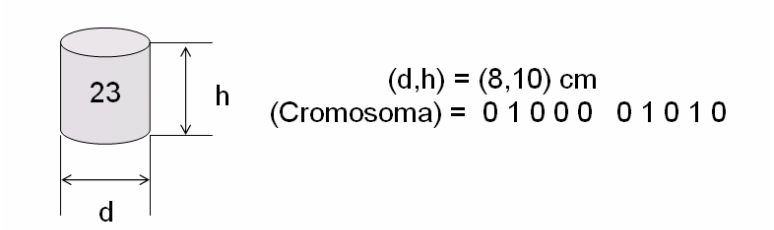


Figura A.2 Una Lata Típica y la Representación de su Cromosoma con Costo Igual 23 Unidades.

Escogiendo el límite bajo y alto como se describió anteriormente se permite al AG considerar solamente valores enteros en el rango [0, 31], sin embargo los AGs no están restringidos para usar valores enteros, el hecho es que los AGs pueden ser asignados para usar algún otro valor entero o no entero justamente cambiando la longitud de la cadena y los límites bajo y alto.

A.5.2 Asignando la Función Objetivo a una Solución (Fitness).

Es importante reiterar que el algoritmo genético binario trabaja con cadenas que representan las variables de decisión, en vez de las mismas variables de decisión. Una vez que una cadena (o una solución) es creada por un operador genético, es necesario evaluar la solución, particularmente en el contexto de la función objetivo y función de restricción. En caso de no haber restricciones, el fitness de una cadena es asignado a un valor que está en función del valor de la solución en la función objetivo. En la mayoría de los casos, sin embargo el fitness se hace igual al valor de la función objetivo. Por ejemplo, el fitness de la lata representada por una cadena "s" de 10 bits es:

$$F(s) = 0.065\left[\pi(8)^2 / 2 + \pi(8)(10)\right] = 23 \qquad \text{(A.2)}$$

Asumiendo que c = 0.065. Dado que el objetivo para la optimización aquí es minimizar la función objetivo, se notará que una solución con un valor de fitness pequeño comparado con otra solución es mejor. La Figura A.3, muestra el fenotipo de una población aleatoria de seis latas. El fitness de cada lata es marcado. Es interesante marcar que dos soluciones no tienen un volumen de 300 ml y son penalizados agregando un costo artificial extra. Actualmente será suficiente notar que el costo penalizado extra es lo suficientemente grande para causar que todas las soluciones impracticas tengan valores de fitness peores que los que están en alguna solución práctica.

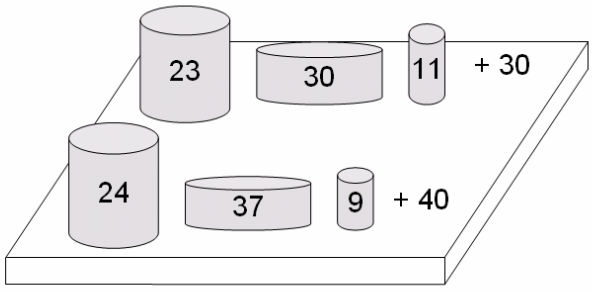


Figura A.3 Una Población Aleatoria de Seis Latas.

A.5.3 Operador de Selección o Reproducción.

El objetivo primario de un operador de reproducción es duplicar las soluciones buenas y eliminar las malas soluciones en una población, mientras se mantiene la población de tamaño constante. Esto se logra ejecutando las siguientes tareas.

1. Identificar las buenas soluciones en una población.
2. Hacer múltiples copias de las buenas soluciones.
3. Eliminar malas soluciones de la población y por lo tanto múltiples copias de las buenas soluciones deben de ser colocadas en la población.

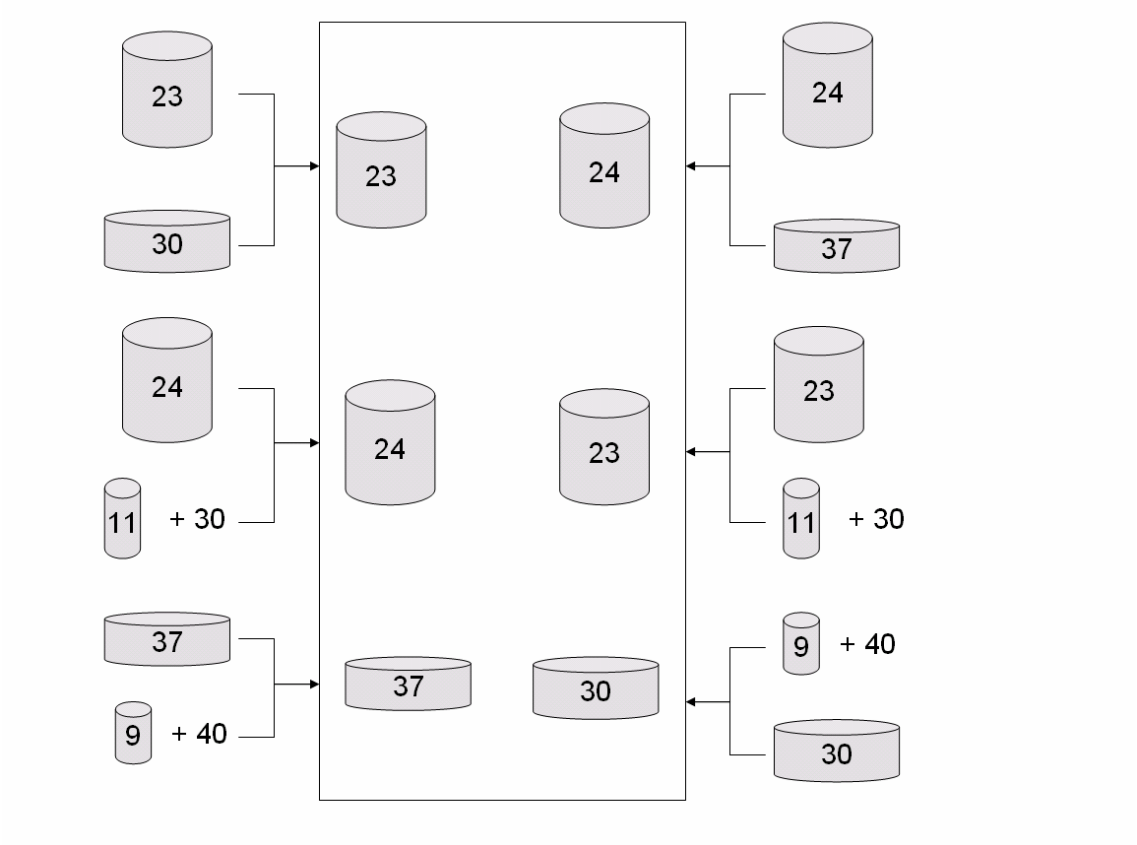


Figura A.4 Un Torneo que es Jugado Entre Seis Miembros de la Población.

Existe un número de formas para lograr las tareas anteriores. Algunos métodos comunes son la selección por torneo, selección proporcional y selección por clasificación [Goldberg 91].

Selección por Torneo. Los torneos son jugados entre dos soluciones y la mejor solución es escogida y colocada en un área de reproducción (mating pool). Otras dos soluciones son tomadas nuevamente y otro slot del área de reproducción es llenado, cada solución puede participar en exactamente dos torneos. La mejor solución en una población ganara ambas veces, más haya de hacer dos copias a la nueva solución. Usando un argumento similar, la peor solución perderá en ambos torneos y será eliminada de la población. En esta forma, alguna solución en la población tendrá cero, una o dos copias en la nueva población. Se ha mostrado que la selección de torneo tiene mejor o equivalente convergencia y complejidad de tiempo computacional cuando es comparado con algún otro operador de reproducción que existe en la literatura [Goldberg 91].

La Figura A.4 muestra seis diferentes torneos jugados entre los miembros de la población vieja (cada uno obtiene exactamente dos torneos). Cuando las latas con un costo de 23 unidades y 30 unidades son escogidas en forma aleatoria para el primer torneo, la lata consistiendo de 23 unidades gana y se copia en el área de reproducción. Las próximas dos latas son escogidas por el segundo torneo y la mejor lata es colocada en área de reproducción nuevamente. Así es como el área de reproducción es formada (ver Figura A.5). Es interesante notar como las mejores soluciones (tienen menos costo) y se han obtenido múltiples copias en el área de reproducción y las peores soluciones han sido descartadas esta es precisamente la propuesta de una reproducción u operador de selección.

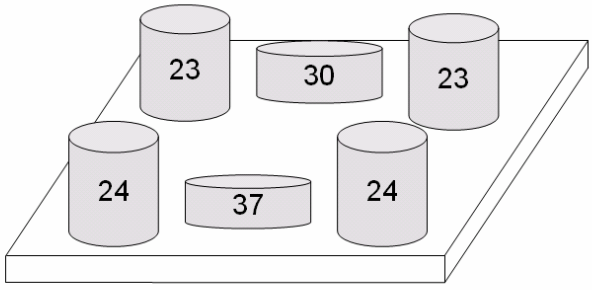


Figura A.5 La Población Después de un Operador de Reproducción.

Selección Proporcional. Las soluciones son copias asignadas, el número es proporcional al valor de la función objetivo. Si la función objetivo promedio de todos los miembros de la población es f_{avg} una solución con una función objetivo f_i obtiene

un número esperado de copias f_i / f_{avg}. Existen algunas variantes de esta técnica pero la más común es por ruleta.

La Ruleta (RWS). La implementación de este operador de selección puede ser pensada como un mecanismo de ruleta, donde la rueda es dividida en N divisiones (tamaño de la población), donde el tamaño de cada uno es marcado en proporción al valor de la función objetivo de cada miembro de la población. Después la rueda gira N veces, cada vez se escoge la solución indicada por el apuntador. La Figura A.6 muestra una ruleta con 5 individuos teniendo diferentes valores en su función objetivo.

Solución, i	Fi	pi	Pi	piN
1	25.0	0.25	0.25	1.25
2	5.0	0.05	0.30	0.25
3	40.0	0.40	0.70	2.00
4	10.0	0.10	0.80	0.50
5	20.0	0.20	1.00	1.00

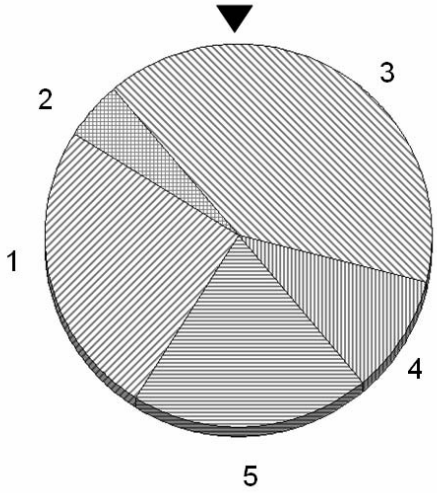


Figura A.6 Una Ruleta con Cinco Individuos de Acuerdo a su Valor en la Función Objetivo. El Tercer Individuo Tiene Mayor Probabilidad de Seleccionarse.

Dado que el tercer individuo tiene más alto valor en su función objetivo que los otros, se espera que la selección de ruleta (RWS) seleccione la tercera solución más a menudo que alguna otra solución. Este esquema de RWS puede ser simulado fácilmente sobre una computadora. Usando el valor de la función objetivo F_i de todas las cadenas, la probabilidad de seleccionar la i^{th} cadena es $p_i = F_i / \sum_{j=1}^{N} F_j$. Después la probabilidad acumulada $\left(P_i = \sum_{j=1}^{N} p_j\right)$ de cada cadena puede ser calculada agregando las probabilidades individuales del tope de la lista. Esto es la cadena de más abajo en la población tiene una probabilidad acumulada (P_N) igual a 1. El concepto de ruleta puede ser simulado haciendo que la i^{th} cadena en la población represente los valores de probabilidad acumulada en el rango $[P_{i-1}, P_i]$. La primer cadena representa el valor

acumulado de cero a P_1. La forma de escoger N cadenas, N números aleatorios entre cero y uno son creados. Esto es, una cadena que representa un número aleatorio escogido en el rango de probabilidad acumulada (calculado del valor de su función objetivo) para la cadena es copiado a un área de apareamiento (meeting pool).

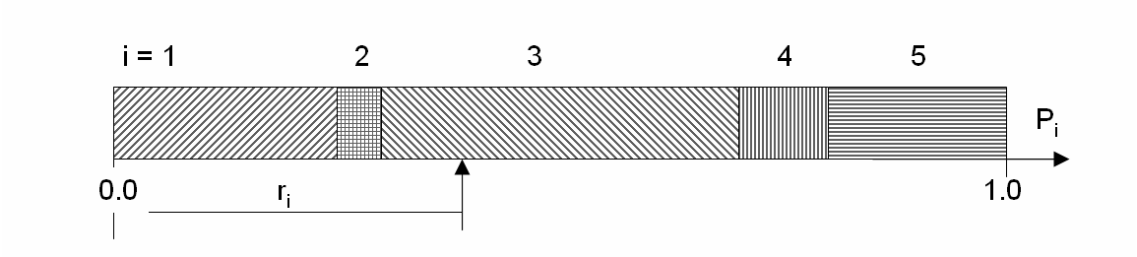


Figura A.7 Una Implementación del Operador de Selección de Ruleta (RWS).

La Figura A.7 muestra la línea de probabilidad acumulada (en un rango de cero a uno) y el correspondiente rango de cada una de las cinco soluciones para el problema mencionado en la Figura A.6. Un número aleatorio r_i mostrado por el apuntador marca la solución 3. La solución 3 es seleccionada como un miembro de la reproducción. De esta forma la cadena con un valor de su función objetivo alto representa un gran rango de probabilidad acumulada y por lo tanto tiene alta probabilidad de ser copiado en la reproducción. Dado que la computación del promedio de los valores de sus funciones objetivo, requieren los valores de las funciones objetivo de todos los miembros de la población, este operador de selección es más lento comparado con el método de selección de torneo [Goldberg 91]. El operador de selección de ruleta inherentemente maximiza la función objetivo. Esta técnica puede implementarse de la manera siguiente.

Algoritmo RWS.

1. *Asume el valor esperado total de los individuos en la población. Llamar a esta suma T.*
2. *Repetir N veces:*
   Escoger un número aleatorio r entre 0 y T.
   Ciclar a través de los individuos de la población, sumando los valores esperados, hasta que la suma sea mayor o igual a r. El individuo cuyo valor esperado haga que esta suma exceda este límite es seleccionado.

Selección por Jerarquía. Es una técnica alternativa cuyo propósito es también prevenir una convergencia demasiado rápida. En la versión propuesta por [Baker 85], los individuos de la población se ordenan de mejor a peor (en base a su aptitud), y cada uno se selecciona tanta veces como corresponda de acuerdo a una función de asignación no incremental; posteriormente, se efectúa selección proporcional de acuerdo a dicha asignación. No hay necesidad de escalar las aptitudes en éste caso. Esto trae ventajas y desventajas. Por ejemplo, podemos prevenir la convergencia prematura al evitar usar las aptitudes reales (ventaja), pero no sabremos qué tanto supera un individuo a otro en términos de aptitud (desventaja). Esta técnica evita que, un pequeño grupo de individuos sea seleccionado demasiadas veces, reduciendo la presión de selección cuando la varianza de aptitud es baja, pues las diferencias entre las aptitudes absolutas no afecta a la jerarquización.

Elitismo. Por primera vez introducido por De Jong en su Tesis Doctoral, establece que el mejor individuo o los mejores de la población sobrevivan de generación en generación. La estrategia de elitismo básica copia el mejor individuo de la población actual a la próxima si éste individuo no ha sido transferido a través del proceso normal de selección, cruzamiento y mutación [Goldberg 89].

A.5.4 Operador de Cruzamiento.

Un operador de cruzamiento es aplicado a las cadenas de reproducción. El operador de reproducción no puede crear nuevas soluciones en la población. Este solo hace más copias de buenas soluciones que remplazan las malas soluciones. La creación de nuevas soluciones son generadas por un operador de mutación y de cruzamiento. Así como en el operador de reproducción, existe un número de operadores de cruzamiento en la literatura de los algoritmos genéticos [Spears 98], pero en casi todos los operadores de cruce, dos cadenas son tomadas de la reproducción en forma aleatoria y alguna porción de la cadena es intercambiada para crear nuevas cadenas. Un operador de cruce en un punto, es ejecutado en forma aleatoria escogiendo un lugar de cruzamiento en las cadenas e intercambiando todos los bits del lugar de cruzamiento.

Ilustramos el operador de cruzamiento para tomar dos soluciones (llamadas soluciones padres) de la nueva población creada después del operador de reproducción. Las latas y

sus genotipos (cadenas) son mostrados en la Figura A.8. El tercer sitio a lo largo de la cadena es seleccionado en forma aleatoria y el contenido del lado derecho de este lugar de cruzamiento es intercambiado entre las dos cadenas. El proceso crea dos nuevas cadenas (llamadas descendientes). Los fenotipos (las latas) son también mostradas en la figura. Dado que un simple lugar de cruzamiento es escogido aquí, el operador de cruzamiento es llamado operador de cruzamiento en un punto.

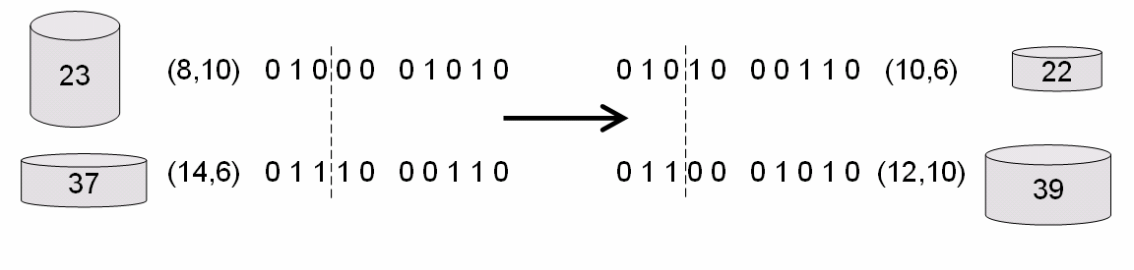


Figura A.8 Operador de Cruzamiento en un Punto.

Es importante notar que el operador de cruzamiento crea una solución (teniendo un costo de 22 unidades) que es mejor que el costo de ambas soluciones padres. Uno pudiera sorprenderse de que si un cruzamiento diferente fuera escogido u otras dos cadenas fueran escogidas para el cruzamiento, tendríamos mejores hijos cada vez. No siempre en el cruzamiento se obtienen mejores hijos que las soluciones padres, pero es claro que la oportunidad de crear mejores soluciones son mejores que las de un número aleatorio.

La forma de preservar algunas buenas cadenas seleccionadas durante el operador de reproducción, no todas las cadenas en la población son usadas en un cruzamiento. Si la probabilidad de cruzamiento de un p_c es usada, entonces $100\,p_c$% cadenas en la población son usados en la operación de cruzamiento y $100(1-p_c)$% de la población son simplemente copiados a la nueva población.

El concepto anterior para intercambiar información parcial entre dos cadenas puede también ser lograda con más de un lugar de cruzamiento. En un operador de cruzamiento en dos puntos, dos diferentes lugares de cruce son escogidos en forma aleatoria. Esto dividirá la cadena en tres sub-cadenas. La operación de cruzamiento es

completada intercambiando la mitad de las sub-cadenas entre las cadenas, como es mostrado en el ejemplo de la Figura A.9:

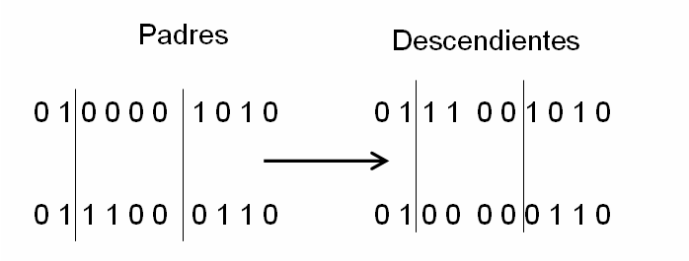


Figura A.9 Operador de Cruzamiento en Dos Puntos.

Extendiendo esta idea, uno puede también implementar, un cruzamiento de n puntos, donde los n cruzamientos son escogidos (impares o parejas). Si un número impar de lugares de cruzamiento es escogido, el final de cada cadena debe ser considerado como un lugar de cruzamiento adicional, más haya de hacer que el número total de lugares de cruzamiento sea un número par. El cruzamiento puede llevarse a cabo intercambiando sub-cadenas alternadas. El extremo de este proceso es un cruzamiento uniforme, donde un descendiente es construido escogiendo cada bit con una probabilidad p (usualmente $p = 0.5$ es usado) de cada padre como se muestra en el ejemplo de la Figura A.10.

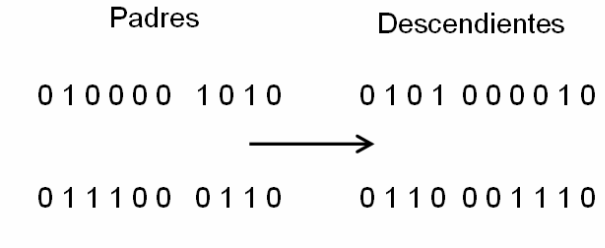


Figura A.10 Operador de Cruzamiento en N Puntos.

En el ejemplo, el segundo, cuarto, quinto, séptimo y noveno bits son intercambiados entre los padres. En términos de extender la potencia de exploración (o búsqueda) de un operador de cruzamiento, un operador de cruzamiento en un punto preserva la estructura de las cadenas padres para entenderse al máximo en los descendientes.

A.5.5 Operador de Mutación.

El operador de cruzamiento es principalmente responsable de los aspectos de búsqueda para un algoritmo genético, pensamos que el operador de mutación es también utilizado para esta propuesta. La mutación de un bit cambia un 1 a un 0 o viceversa, con probabilidad de mutación p_m. La necesidad de mutación es mantener diversidad en la población. La Figura A.11 muestra como una cadena obtenida después de usar los operadores de reproducción y cruzamiento ha sido mutada a otras cadenas, esto representa una ligera diferencia en la lata. Una vez más las solución obtenida en la ilustración es mejor que la solución original. Pensamos que esto no pudiera suceder en todas las instancias de la operación de mutación, alterar una cadena con una pequeña probabilidad no es una operación aleatoria, dado que el proceso puede crear solamente algunas soluciones en el espacio de búsqueda.

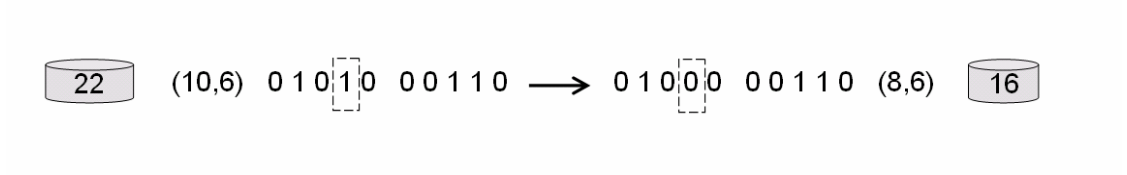


Figura A.11 Operador de Mutación en un Bit.

Los tres operadores: selección, cruzamiento y mutación son simples y concretos. El operador de reproducción selecciona buenas cadenas, mientras que el operador de cruzamiento recombina dos buenas cadenas para ayudar a formar mejores sub-cadenas. El operador de mutación altera una cadena localmente para ayudar a crear una mejor sub-cadena. Dado que ninguna de esas operaciones son ejecutadas determinísticamente, esas peticiones no son garantizadas, nada explícitamente es evaluado, durante la generación de un GA. Sin embargo es de esperarse que si malas cadenas son creadas, serán eliminadas por el operador de reproducción en subsecuentes generaciones, y si buenas cadenas son creadas, permanecerán a través de generaciones.

4.6 Entendiendo como Trabajan los Algoritmos Genéticos (GAs).

Para investigar como trabajan los algoritmos genéticos, aplicamos el AG para observar un ciclo en un problema de maximización numérica.

$$Maximizar[sen(x)], \quad 0 \le x \le \pi \quad \text{variable atada “x”} \tag{A.3}$$

Usaremos cinco cadenas (strings) que representan la variable “x” en un rango de $[0, \pi]$, por lo tanto la cadena (00000) representa la x = 0 y la cadena (11111) representa la solución x = π. Los otras 30 cadenas son mapeadas en un rango de $[0, \pi]$, uniformemente. Asumimos también que utilizamos una población de tamaño 4, selección proporcional, operador de cruzamiento en un punto con $p_c = 1$, y ninguna mutación (o, $p_m = 0$). Para comenzar la simulación de un algoritmo genético, creamos una población inicial aleatoria, evaluamos cada cadena y entonces usamos tres operadores genéticos, como se muestran en la Tabla A.1.

Población inicial

Cadena	DV[a]	x	f(x)	f_i / f_{avg}	AC[b]	Reproducción
01001	9	0.912	0.791	1.390	1	01001
10100	20	2.027	0.898	1.579	2	10100
00001	1	0.101	0.101	0.178	0	10100
11010	26	2.635	0.485	0.853	1	11010
		Promedio, f_{avg} =	0.569			

Reproducción	CS[c]	Nueva Población			
		Cadena	DV[a]	x	f(x)
01001	3	01000	8	0.811	0.725
10100	3	10101	21	2.128	0.849
10100	2	10010	18	1.824	0.968
11010	2	11100	28	2.838	0.299
			Promedio, f_{avg} =		0.710

[a] **DV, decodifica el valor de la cadena**
[b] **AC, valor actual de la cadena en la población**
[c] **CS, cruzamiento**

Tabla A.1 Una Generación Simulada de los AGs para la Función [sen(x)].

La primer cadena tiene un valor codificado igual a 9 y esta cadena corresponde a una solución x = 0.912, que tiene una valor de función igual a sen(0.912) = 0.791. Similarmente, las otras tres cadenas son también evaluadas. Dado que el esquema de reproducción proporcional asigna un número de copias de acuerdo a la función objetivo de la cadena, el número esperado de copias de cada cadena es calculado en la columna 5.

Cuando el operador de selección proporcional es implementado, el número de copias alojadas de la cadena es mostrado en la columna 6. La cadena 7 muestra la reproducción. Es notable que la tercer cadena en la población inicial tiene una función objetivo que es muy pequeña comparada con el promedio de la población y es eliminada por el operador de selección. Por otra parte la segunda cadena, será una buena cadena, se hacen dos copias en el área de reproducción. El lugar de cruzamiento es seleccionado en forma aleatoria y cuatro nuevas cadenas creadas después del cruzamiento son mostradas en la columna en la parte baja de la tabla.

Dado que ninguna mutación es usada, ninguno de los bits son alterados. Esto es la columna 3 de la parte inferior, representa la población al final de un ciclo de un GA. Después, cada una de las cadenas son decodificadas, mapeadas y evaluadas. Esto completa una generación de la simulación de un GA. La función objetivo promedio de la nueva población es obtenida para ser 0.710, que es una función objetivo mejorada en comparación con la población inicial. Es interesante notar que todos los operadores usaron números aleatorios, un algoritmo genético con los tres operadores produce una búsqueda directa, que usualmente resulta en un incremento de las soluciones de una generación a la próxima.

Anexo B. Fundamentos del Filtrado Morfológico.

La idea básica del procesamiento de imágenes morfológicas es que el elemento estructural sea usado para examinar un conjunto de imágenes. El centro del elemento estructural (generalmente un arreglo) es movido de un píxel al próximo en la imagen. Un conjunto de operaciones es usado entre el elemento estructural y la parte de la imagen que es traslapada con este elemento. El conjunto de operaciones produce información estructural acerca de la imagen.

Históricamente, el procesamiento de imágenes morfológicas es con imágenes binarias y procesamiento de imágenes en escala de grises. Solo trabajaremos el caso binario.

Para ejecutar el procesamiento de imágenes morfológicas en dos dimensiones. Usamos un elemento estructurado en dos dimensiones. El elemento estructurado en dos dimensiones se traslada a través de la imagen. El procesamiento de imágenes morfológico utiliza un conjunto de operaciones más que la multiplicación y la suma como lo hacía la convolución. Como resultado, el procesamiento de imágenes morfológico es parte del procesamiento de imágenes no lineal.

En el caso binario el elemento estructural y la imagen tienen solamente dos valores. El elemento es movido sobre una imagen y comparado elemento por elemento. Los dos valores son típicamente (0,1) o (0, 255) o (falso, verdadero). Típicamente el elemento estructurado está formado por un arreglo de dos dimensiones que tienen un origen. El origen es típicamente diseñado como el centro del arreglo. El anexo tiene como objetivo mostrar la dilatación, erosión, apertura y cierre, que implementan la teoría de conjuntos para dar origen al filtrado morfológico, tal como lo describe Douglas A. Lyon [Lyon 99], Rafael C. González y Richard E. Woods [González 95] y Gonzalo Pajares [Pajares 04].

B.1 Teoría de Conjuntos.

Un conjunto es una colección de elementos, por ejemplo, el conjunto de todos los libros en la biblioteca. El número de elementos en un conjunto es llamado cardinalidad. La

cardinalidad es un conjunto que puede ser infinito. La A es un conjunto y la x es un elemento contenido en A. Decimos "x es un miembro del conjunto A" y escribimos

$$x \in A \tag{B.1}$$

Si x no es un miembro del conjunto A, entonces escribimos

$$x \notin A \tag{B.2}$$

Si el elemento x tiene una propiedad , llamada P, entonces escribimos

$$P(x) \tag{B.3}$$

Para describir el conjunto de todos los elementos, x tal que (B.3) es verdadera, escribimos

$$\{x / P(x)\} \tag{B.4}$$

El conjunto vacío o conjunto nulo es el conjunto cuya cardinalidad es cero y esta es denotada como $\varnothing$.

Los conjuntos deben ser coleccionados por otros conjuntos. El hecho, es que la cardinalidad de $(\varnothing, \varnothing)$ es dos. El conjunto de todos los enteros es denotada Z. La cardinalidad de Z es infinita. La cardinalidad de (Z, Z) es dos.

Si el conjunto B esta contenido en A, entonces B es un subconjunto de A y escribimos:

$$B \subseteq A \tag{B.5}$$

Si dos conjuntos tienen los mismos elementos, entonces ellos son iguales y escribimos

$$A = B \tag{B.6}$$

Si dos conjuntos no tienen los mismos elementos, escribimos

$$A \neq B \tag{B.7}$$

La unión de todos los elementos en un conjunto A con un conjunto B forman un nuevo conjunto llamado A unión B: Esto es, el conjunto de todos los elementos que están en A o en B:

$$A \cup B = \{x / x \in Aox \in B\} \tag{B.8}$$

El operador unión puede ser mapeado a todos los conjuntos en un conjunto:

$$\cup\{A, B\} = A \cup B \tag{B.9}$$

La intersección entre los conjuntos A y B producen un nuevo conjunto llamado A intersección B. Este es el conjunto de todos los elementos que están en A y en B.

$$A \cap B = \{x / x \in Aandx \in B\} \tag{B.10}$$

Los operadores de intersección pueden ser mapeados a todos los conjuntos en un conjunto:

$$\cap\{A, B\} = A \cap B \tag{B.11}$$

El complemento de B, relativo a A, produce un nuevo conjunto llamado la diferencia de A y B. Estos son los elementos que están en A pero que no están en B:

$$A[/]B = \{x / x \in Aandx \notin B\} = A \cap B^c = A[/](A \cap B) \tag{B.12}$$

Podemos a hora definir las *Leyes De Morgan* de la siguiente manera:

$$A[/](B \cup C) = (A[/]B) \cap (A[/]C)$$

$$A[/](B \cap C) = (A[/]B) \cup (A[/]C) \quad \text{(B.13)}$$

Alguna literatura usa el símbolo "-", para establecer la diferencia entre conjuntos. Nosotros usamos "[/]" para establecer la diferencia entre conjuntos.

El subconjunto básico es un conjunto que contiene todos los elementos primarios y colecciones de los subconjuntos y es denotado por S.

El complemento de B relativo a S es simplemente llamado el complemento de B y es escrito como:

$$S[/]B = B^c = \{x / x \in S and x \notin B\} \quad \text{(B.14)}$$

Por ejemplo, supóngase que A representa el conjunto de los estudiantes que pasan la clase java. Suponga que B representa el conjunto de los estudiantes que fracasan en la clase java. A unión B es el conjunto de todos los estudiantes que tomaron java y es denotado por S. Posteriormente, A intersección B es el conjunto nulo (dado que un estudiante no puede reprobar ni aprobar). La diferencia de A y B es igual a A por que el conjunto A y B no se interceptan. También el complemento de B es igual al conjunto A. Como un resultado, decimos que el conjunto A y B es mutuamente exclusivo.

B.2 Dilatación.

La transformada morfológica de la dilatación $\oplus$ combina dos conjuntos utilizando la adición de vectores (o adición de conjuntos de Minkowski). La dilatación $A \oplus B$ es el conjunto de puntos de todas las posibles adiciones vectoriales de pares de elementos, uno de cada conjunto X y B.

$$X \oplus B = \{d \in E^2 : d = x + b\} \text{ para cada } x \in X \text{ y } b \in B \quad \text{(B.15)}$$

Enseguida mostramos un ejemplo de dilatación.

$$X = \{(0,1),(1,2),(2,0),(2,1),(3,0),(3,1)\}$$
$$B = \{(0,0),(0,1)\}$$
$$X \oplus B = \{(0,1),(1,2),(2,0),(2,1),(3,0),(3,1),(0,2),(1,3),(2,2),(3,2)\}$$

*0	1	0	0	0					*0	1	1	0	0
0	0	1	0	0					0	0	1	1	0
1	1	0	0	0	$\oplus$	*1	1	=	1	1	1	0	0
1	1	0	0	0					1	1	1	0	0
0	0	0	0	0					0	0	0	0	0

Tal como observamos en el ejemplo anterior una transformación morfológica viene dada por la relación de la imagen (conjuntos de puntos X) con otro pequeño conjunto de puntos B, llamado elemento estructural.

Gráficamente, la dilatación se realiza como sigue: se va recorriendo la imagen por ejemplo de izquierda a derecha y de arriba abajo y, donde nos encontremos un 1, situamos el origen del elemento estructural sobre ese 1; en esa posición se realiza la unión del elemento estructural con la parte de la imagen sobre la que se solapa dicho elemento; si todos los unos del elemento estructural coinciden con los unos de la imagen, entonces marcamos el píxel de la imagen donde está el origen del elemento estructural con el valor lógico de 1.

B.3 Erosión.

La transformación morfológica de la erosión $\otimes$ combina dos conjuntos utilizando la sustracción de vectores. Es dual de la dilatación. Ni la erosión ni la dilatación son transformaciones invertibles:

$$X \otimes B = \{d \in E^2 : d + b \in X\} \text{ para cada } \{b \in B\} \qquad \text{(B.16)}$$

Esta expresión dice que cada punto d del conjunto X, que para nosotros será la imagen, es comprobado; el resultado de la erosión está dado por los puntos d para los cuales

todos los posibles d + b están en X. A continuación mostramos un ejemplo del conjunto de puntos X erosionados por el elemento estructural B:

$$X = \{(0,2),(1,2),(2,0),(2,1),(2,2),(2,3),(3,2),(4,2)\}$$
$$B = \{(0,0),(0,1)\}$$
$$X \otimes B = \{(2,0),(2,1),(2,2)\}$$

*0	0	1	0	0				*0	0	0	0	0
0	0	1	0	0				0	0	0	0	0
1	1	1	1	0	⊗	*1	1	=	1	1	1	0	0
0	0	1	0	0				0	0	0	0	0
0	0	1	0	0				0	0	0	0	0

Gráficamente, la erosión se realiza como sigue: se va recorriendo la imagen, por ejemplo, de izquierda a derecha y de arriba abajo; allí donde nos encontremos un 1, situamos el origen del elemento estructural sobre ese 1; si todos los unos del elemento estructural coinciden con los unos de la imagen, entonces marcamos el píxel de la imagen donde está el origen del elemento estructural con el valor 1.

B.4 Apertura y Cierre.

La erosión y dilatación son transformaciones no invertibles. Si una imagen es erosionada y luego dilatada, la imagen original no se recupera. En efecto, el resultado es una imagen más simplificada y menos detallada que la imagen original.

La erosión seguida de una dilatación crea una transformación morfológica importante llamada *apertura*. La *apertura* de una imagen X por un elemento estructural B se denota por X o B y se define como:

$$X \circ B = (X \otimes B) \oplus B \qquad \text{(B.17)}$$

La dilatación seguida de una erosión crea una transformación morfológica llamada cierre. El cierre de una imagen X por un elemento estructural B se denota por $X \bullet B$ y se define como:

$$X \bullet B = \left(X \oplus B \right) \otimes B \qquad \text{(B.18)}$$

Si una imagen X permanece invariable por apertura con respecto al elemento estructural B, se dice que es *abierta* con respecto a B. Análogamente, si una imagen X permanece invariable por cierre con respecto al elemento estructural B, se dice que es *cerrada* con respecto a B.

La apertura y el cierre con un elemento estructural isótropo se utiliza para eliminar detalles específicos de la imagen más pequeños que el elemento estructural. La forma global de los objetos no se distorsiona. El cierre conecta objetos que están próximos entre sí, rellena pequeños huecos y suaviza el contorno del objeto rellenando los pequeños valles mientras que la apertura produce el efecto contrario. Los conceptos de pequeño y próximo están relacionados con la forma del elemento estructura.

Bibliografía.

[Arapé 03] 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.

[Back 97] Back T. Handbook of Evolutionary Computation, pp. C7.1:1-15. Bristol: Institute of Physics Publishing and New York: Oxford University Press 1997.

[Baker 85] Baker J. E. Adaptive selection methods for genetic algorithms. In proceedings of an international conference on genetic algorithms and their applications, pp. 101-111.

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

[Bigus 01] Bigus P. Joseph and Bigus jennifer, Constructing Intelligent Agents with Java. A Programmer´s Guide to Smarter Applications. John Wiley & Sons, Inc. 2001.

[Booch 94] Booch, Grady. Object oriented analysis and design with applications. Benjamin Cummings, (1994).

[Booch 99] Booch Grady, Rumbaugh James y Jacobson Ivar. El Lenguaje Unificado de Modelado. Addison Wesley 1999.

[Bryant 81] Bryant R. M. and Finkel R. A. A Stable Distrbuted Scheduling Algorithm. In Proceedings of the 2nd International Conference on Distributed Computing Sysems, 314-23.

[Carretero 01] Carretero Pèrez J., Garcìa Carballeira F., De Miguel Anasagasti P. y Pérez Costoya

F. SISTEMAS OPERATIVOS Una Visión aplicada, McGraw-Hill, Madrid, España 2001.

[Casavant 88a] Casavant T. L. and Kuhl J. G. A Taxonomy of Scheduling in General Purpose Distributed Computing Systems. IEEE Transactions on Software Engineering, SE-14(11), 1578-88.

[Eager 86] Eager D. L., Lazowska and Zahorjan J. A Comparison of Receiver Initiated and Sender-Initiated Adaptive Load Sharing. Performance Evaluation, 6(1), 53-68.

[Eager 88] Eager D. L., Lazowska and Zahorjan J. The Limited Performance Benefits of Migrating Active Processes for Load Sharing. In Proceedings of the 1988 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pp. 63-72, Santa Fe, New Mexico.

[Ferber 99] Ferber Jacques. Multi-Agent Systems. An Introduction to Distributed Artificial Intelligence, Addison-Wesley, 1999.

[Fowler 97] Fowler Martin con Scott Kendall. UML Gota a Gota. Addison Wesley 1997.

[Gen 97] Gen M. and Cheng B. Genetic Algorithms and Engineering Design, New York, Wiley 1997.

[Goldberg 89] Goldberg D. E. Genetic Algorithms for Search, Optimization, and Machine Learning. Reading, MA: Addison-Wesley 1989.

[Goldberg 91] Goldberg D. E. and Deb K. A Comparison of Selection Schemes Used in Genetic Algorithms. In Foundations of Genetic Algorithms 1.(FOGA-1), pp. 69-93.

[González 96] González C. Rafael y Woods E. Richard. Tratamiento Digital de Imágenes. Addison-Wesley 1996.

[Goscinski 94] Goscinski, Andrzej, Distributed Operating Systems. The Logical Design. Addison-

Wesley 1992.

[Hilera 00] Hilera R. José y Martínez J. Victor, Redes Neuronales Artificiales (Fundamentos, Modelos y Aplicaciones). Alfaomega & Ra-Ma, 2000.

[Haupt 98] Haupt Randy L. and Haupt Sue E. Practical Genetic Algorithms. Wiley Inter-Science 1998.

[Hillier 02] Hillier Frederick S., Lieberman Gerald J. Investigación de Operaciones. Septima Edición, McGraw-Hill 2002.

[Holland 75] Holland J. H. Adaptation in Natural and Artificial Systems. Ann Arbor, MI: MIT Press 1975.

[Hui 99] Hui C. And Chanson T. S. "Improved Strategies for Dynamic Load Balancing " IEEE Concurrency, vol. 7, July 1999.

[IONA 02] IONA Technologies, Tri-Pacific Software Inc, VERTEL Corporation, Load Balancing and Monitoring, Supported by Alcatel, Institut 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.

[Justino 01] Justino, E. J. R., Bortolozi, F., and Sabourin, R. Off-line signature verification using HMM for random, simple and skilled forgeries. Proceedings of the Sixth International Conference on Document Analysis and Recognition, 2001. 1[1], 1031-1034. 10-9-2001. Seattle, WA, USA., IEEE.

[Kalyanmoy 01] Kalyanmoy Deb. Multi-Objective Optimization Using Evolutionary Algorithms. First Edition, Wiley 2001.

[Kameda 97] Hisao Kameda, Jie Li, Chonggun Kim and Yongbing Zhang, Optimal Load

Balancing in Distributed Computer System. Springer-Verlag, London 1997.

[Kidwell 94] Kidwell M. D. and Cook D. J. "Genetic Algorithm for Dynamic Task Scheduling", Proc. IEEE 13th Ann. Int'l Phoenix Conf. Computers and Comm., pp. 61-67, 1994.

[Koza 92] Koza, J. R. Genetic Programming: On the programming of computer by means of natural selection. Cambridge, MA: MIT Press.

[Krueger 84] Krueger, P. and R. Finkel. "An Adaptive Load Balancing Algorithm for a Multicomputer. Technical Report 539, University of Wisconsin-Madison, Apr. 1984.

[Krueger 87] Krueger P. and Livny M. Load Balancing, Load Scharing and Performance in Distributed Systems. Computer Science Technical Report #700, Computer Science Department, University of Wisconsin-Madison.

[Krueger 94] Krueger Phillip and Shivaratri G. Niranjan, Adaptive Location Policies for Global Scheduling, IEEE Transactions on Software Engineering, Vol. 20, No.6, June 1994.

[Lee 96] Lee, L. L. and Lizárraga, M.-G. An Off-Line Method for Human Signature Verification. IEEE. 3[1], 195-198. 1996. N.Y., USA, IEEE. Proceedings of the 13th International Conference on Pattern Recognition, 1996. IEEE.

[Lindermeier 00] 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

[Lin 87] Lin H. C. Frank and Séller M. Robert. The Gradient Model Load Balancing Method. IEEE Transactions on Software Engineering, Vol. SE-13, No. 1, January 1987.

[López 01] López Gómez Genoveva, Salas Blanco Mónica, Siles Peláez Raúl y Soriano Campo Fco. Javier. Arquitectura de Objetos Distribuidos CORBA, Facultad

de Informática de Madrid, Abril del 2001.

[Lyon 99] Lyon A. Douglas. Image Processing in JAVA. Prentice Hall 1999.

[Maravall 93] Maravall Gómez-Allende Darío. Reconocimiento de Formas y Visión Artificial. Addison Wesley 1993.

[Martínez 04] Martínez Romo Julio Cesar. Verificación de Firmas Manuscritas en Línea con Modelado Optimo de Características y Aproximación Digital Forense. Tesis de Doctorado en Ingeniería Eléctrica, DEPFI-UNAM, Octubre del 2004.

[Mirchandaney 89] 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.

[Michalewicz 92] Michalewicz Z. Genetic Algorithm + Data Structures = Evolution Programs. Berlin:Springer-Verlag 1992.

[Milenkovic98] Milenkovic Milan. Sistemas Operativos CONCEPTOS Y DISEÑO, segunda edición, McGraw-Hill/Interamericana de España, S. A. U. 1998.

[Muñiz 95] Muñiz, F. J. and Zaluska E. J. Paralled load-balancing: An extension to the gradient model. Parallel Computing 21(1995) 287-301

[Narendra 74] Narendra K. S. and Thathacher M. A. L. Learning Automata. A survey. IEEE Transactions on Systems, Man, and Cybernetics, SMC-4, July.

[Obitko 98] Obitko Marek. Genetic Algorithms. http://cs.felk.cvut.cz/~xobitko/ga/.

[OMG 99] Object Management Group. Fault Tolerant CORBA Specification, OMG Document orbos/99-12-08 ed., December 1999.

[OMG 01] OMG (Object Management Group). The Common Object Request Broker: Architecture and Specification. Technical Report, http:// www.omg.org(2001).

[OMG 01a] Object Management Group, Load Balancing and Monitoring for CORBA-based Applications, Request for Proposal, Tech. Rep., OMG Document (orbos/2001-04-27), Apr. 2001

[OMG 02] Common Object Request Broker Architecture: Core Specification Version 3.0. November 2002 .
http://www.omg.org/technology/documents/corba_spec_catalog.htm.

[Orfali 98] Orfali Robert and Harkey Dan. Client/Server Programming With CORBA and JAVA, Second Edition, Wiley 1998.

[Ossama 01a] 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.

[Ossama 01b] Ossama Othman, O´Ryan Carlos and Schmidt. Designing an Adaptive CORBA Load Balancing Service Using TAO. "IEEE Distributed Systems on Line", Vol. 2, Apr. 2001.

[Ossama 01c] Ossama Othman and Douglas C. Schmidt, Optimizing Distributed System Performance via Adaptive Load Balancing, ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001), Snowbird, Utah, June 18, 2001.

[Ossama 03a] Ossama Othman, Balasubramania Jaiganesh and Douglas C. Schmidt, "The Performance Evaluation of an Adaptive Middleware Load Balancing and Monitoring Service", submitted to the 24th IEEE International Conference on Distributed Computing Systems (ICDCS), May 23-26, 2004, Tokyo Japan.

[Ossama 03b] Ossama Othman, Balasubramania Jaiganesh and Douglas C. Schmidt, "The Design of an Adaptive Middleware Load Balancing and Monitoring Service", in LNCS/LNAI: Proceedings of the Third International Workshop on Self-Adaptive Software, Heidelberg, June

2003, Springer-Verlag.

[Pajares 04] Pajares Gonzalo, de la Cruz M. Jesús, Molina M. José y López Alejandro. Imágenes Digitales, Procesamiento Práctico con Java. Alfaomega 2004.

[Plamondon 00] Plamondon, R. and Shihari, S. N., "Online and off-line handwritting recognition: a comprehensive survey.," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 1, pp. 63-84, Jan.2000.

[Russell 95] Russell S. J., Norvig P. Artificial Intelligence: A Modern Approach, Prentice-Hall, Englewood Cliffs, NJ, 1995.

[Silberschatz 04] Silberschatz A., Galvin B. P., and Gagne G. OPERATING SYSTEM CONCEPTS with JAVA. Sixth Edition, Wiley 2004.

[Scallan 00] Scallan Todd, Monitoring and Diagnostics of CORBA Systems. Demystifying the CORBA Communication Bus to Enable ´Distributed Debugging´. Java Developers Journal. June-2000.

[Schmidt 00] Schmidt, D. C., Stal M., Rohnert H. and Buschmann, F. Pattern- Oriented Software Architecture: Patterns for Concurrency and Distributed Objects, Volume 2. New York, NY: Wiley & Sons, 2000.

[Shirazi 95] Shirazi A. Behrooz, Hurson R. Ali and Kavi M. Krishna. Sheduling and Load balancing in Parallel and Distributed Systems. IEEE Computers Society Press, Los Alamitos, California USA, 1995.

[Shivaratri 90] Shivaratri, N. G., and P. Krueger, “Two Adaptive Location Policies for Global Scheduling”, Proceedings of the 10th International Conference on Distributed Computing Systems, May 1990, pp. 502-509.

[Singhal 94] Singhal Muskesh and Shivaratri G. Niranjan, Advanced Concepts in Operating Systems (Distributed, Database and Multiprocessor Operating Systems). McGraw-Hill 1994.

[Taha 92] Taha Hamdy A. Investigación de Operaciones. Quinta Edición, Alfaomega 1992.

[Vallejos 01] Vallejos Pacheco Carol. OBJETOS DISTRIBUIDOS CON CORBA "Una Introducción a CORBA".
http://www.umsanet.edu.bo/docentes/jhreyes/seminario101/corba/arqcorba.htm.

[Vogel 98] Vogel andreas, Duddy Keith. JAVA Programming with CORBA. Second Edition, Wiley Computer Publishing.

[Vose 99] Vose M. D. Simple Genetic Algorithm: Foundation and Theory. Ann Arbor, MI: MIT-Press 1999.

[Walpole 99] Walpole Ronaldo E., Myers Raymond H. y Myers Sharon L. Probabilidad y Estadística para Ingenieros. Sexta Edición, Addison Wesley 1999.

[Yen 99] Yen John, Langari Reza. FUZZY LOGIC, Intelligence, Control and Information. Prentice Hall, 1999.
[Yu 04] 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.

[Zomaya 01] Zomaya Y. Albert and The Yee-Hwei. Observations on Using Genetic Algorithms for Dynamic Load-Balancing. IEEE Transactions on Parallel and Distributed Systems, Vol 12. No. 9, September 2001.

Publicaciones.

- Máquina Virtual para el Control Distribuido de Objetos en Plataformas Heterogéneas. Encuentro Nacional de Computación (ENC'99), Sep de 1999, Pachuca Hidalgo, Mex.

- Shell para el Procesamiento Distribuido de Imágenes (un Enfoque Heterogéneo). Congreso Internacional de Computación (CIC2000-IPN), efectuado en la ciudad de México, D.F. del 15 al 17 de Noviembre del 2000.

- Balanceo Dinámico de Carga con Agentes Móviles. Tercer Congreso Internacional Sobre Ingeniería Eléctrica-Electrónica, Celebrado en Noviembre del 2002 en la Cd. de Aguascalientes, Ags, México.

- Aplicando un Algoritmo Genético para Balancear Carga Dinámicamente en Ambientes Distribuidos Orientados a Objetos (CORBA). Congreso Mexicano de Computación Evolutiva (COMCEV´03-CIMAT). Celebrado del 28 al 30 de mayo del 2003, en la Cd. de Guanajuato México.

- Scheduling Genetic Strategies for Dynamic Load Balancing in CORBA. Congreso Internacional Sobre Ingeniería Eléctrica y Electrónica (CIIIEE04), Celebrado en Noviembre del 2004 en la Cd. de Aguascalientes, Ags, México.

- Combinando Least_Loaded Más una Estrategia Genética para Mejorar el Balanceo Dinámico de Carga en CORBA (un Enfoque en el Algoritmo Genético). II Congreso Mexicano de Computación Evolutiva (COMCEV´05-UAA). Celebrado del 25 al 27 de mayo del 2005, en Universidad Autónoma de Aguascalientes, Ags México.

- 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.

- Combining Genetic Strategy with Least-Loaded to Improve Dynamic Load Balancing Under CORBA in an Off-line Signature Verification System with Neural Networks Classifier. The Editorial Office of Journal of Supercomputing, 101 Philip Drive, Assinippi Park, Norwell, MA 02061, U.S.A. January 2006.

- Improving Least-Loaded with Genetic Algorithms for Dynamic Load Balancing in CORBA. The Editorial Office of Journal of Parallel and Distributed Computing (ElSEVIER-JPDC) 525 B Street, Suite 1900 San Diego, CA 92101-4495, U.S.A. May 2006.

Articulo Donde Intervine Como Revisor.

- Adel Torkaman Rahmani, Vahid Rafe. Providing Local ORB-Like Services to Collocated CORBA Components. Manuscript which has been submitted to Journal of Supercomputing. The Editorial Office of Journal of Supercomputing, 101 Philip Drive, Assinippi Park, Norwell, MA 02061, U.S.A. Octubre 14, del 2005.

Trabajo a Futuro.

Hasta este momento hemos considerado el balanceo de carga local, es decir considerando solo el balanceo dinámico de carga en una red de área local (Local Area Network (LAN)), observamos como podemos mejorar LL usando una estrategia genética en combinación con LL, también observamos como en el análisis comparativo nuestra estrategia se desempeño mejor bajo ciertas condiciones del sistema, pero la pregunta sería, ¿Que sucede cuando existe más de una LAN?, la respuesta es, que 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 utilizaremos un modelo de equilibrio de carga basado en lógica borrosa (Fuzzy-Logic) [Yen 99], en conjunto con el algoritmo distribuido Request for Bids [Ferber 99].

Balanceando la Carga en Sistemas Distribuidos a Gran Escala Utilizando un Modelo Fuzzy-Logic.

En esta sección se 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-Logic, esos valores son comparados y el mínimo de los dos es proyectado sobre la función de membresía de su gráfica consecuente. El proceso defusificación genera un valor centroide que representa el rango de la calida del servicio (Quality of Service (QoS)). El manager puede entonces enviar los requerimientos del cliente al mejor bidder basándose en la QoS obtenida para cada bidder.

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 [Yen 99, Yu 04].

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 $F = \{(u, u_F(u)) / u \in U\}$ 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.

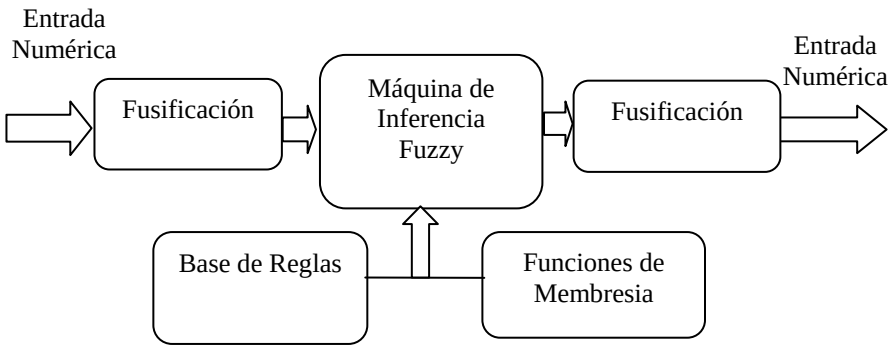


Figure 1 Arquitectura del Modelo Fuzzy-Logic.

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.

Defusicación. 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.

Algoritmo Distribuido Request for Bids.

Generalmente un balanceador de carga debe 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 [Ferber 99], es el algoritmo que se implementará en nuestro balanceador para equilibrar la carga. El algoritmo tiene cuatro estados:

1). El primer estado es dedicado a los Request for Bids mismos. El manejador (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 de los bids y se premiara a otro bidder (y se comienza nuevamente en 3).

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 [Mirchandaney 89]. El manager necesita usar razonamiento aproximado para manejar información fuzzy que

haga eficiente el sistema [Yen 99, Yu 04]. Para hacer una decisión de balanceo correcta, las variables lingüísticas de la carga de los demás bidders (CargaBidder), el tiempo de invocación de métodos remotos (RMIT) y la calidad de servicio (Quality of Service) es definida en el modelo fuzzy de la siguiente manera:

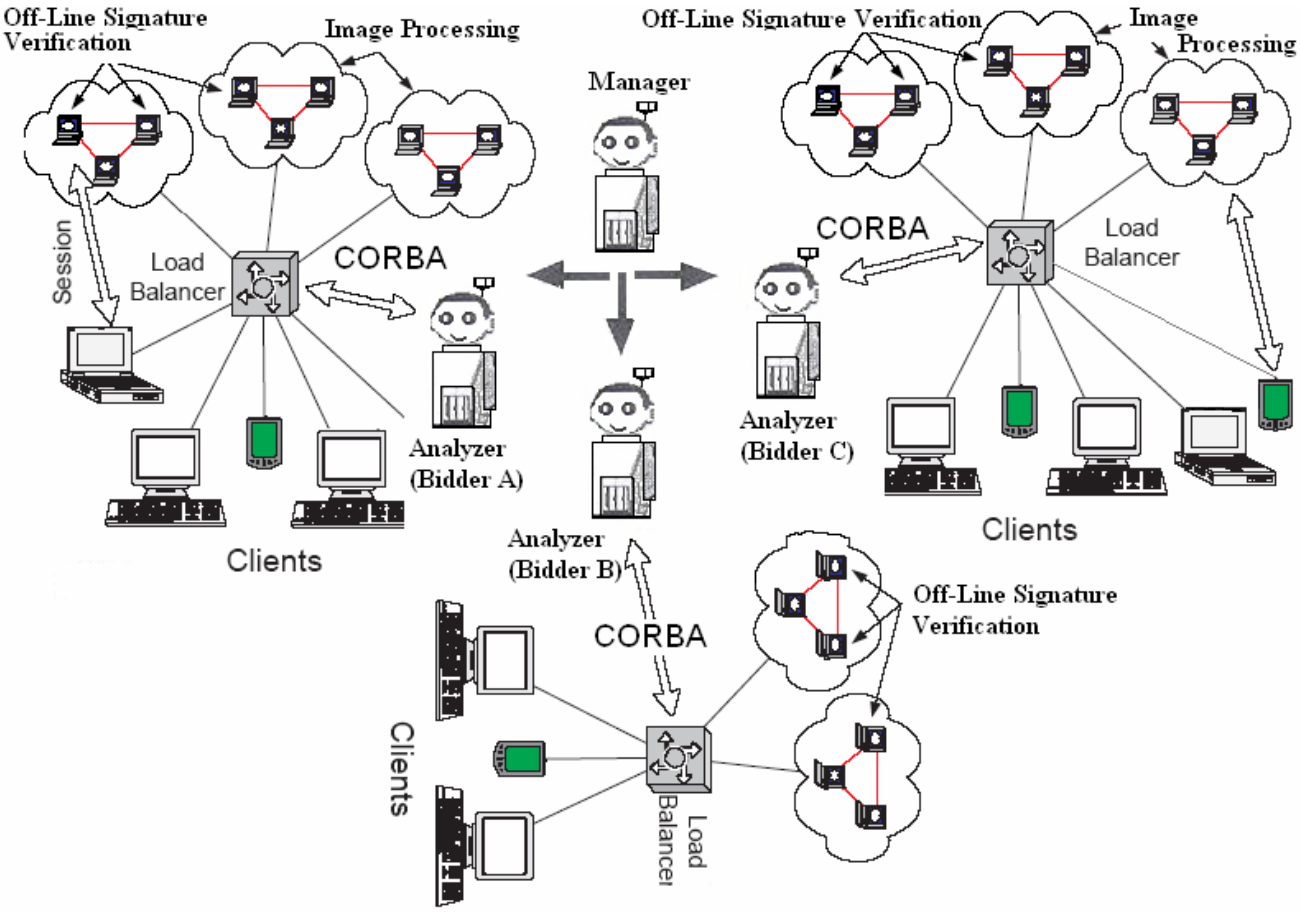


Figura 2 Algoritmo Request for Bids para Balancear Dinámicamente la Carga Entre Varios Bidders.

Carga Bidder.

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

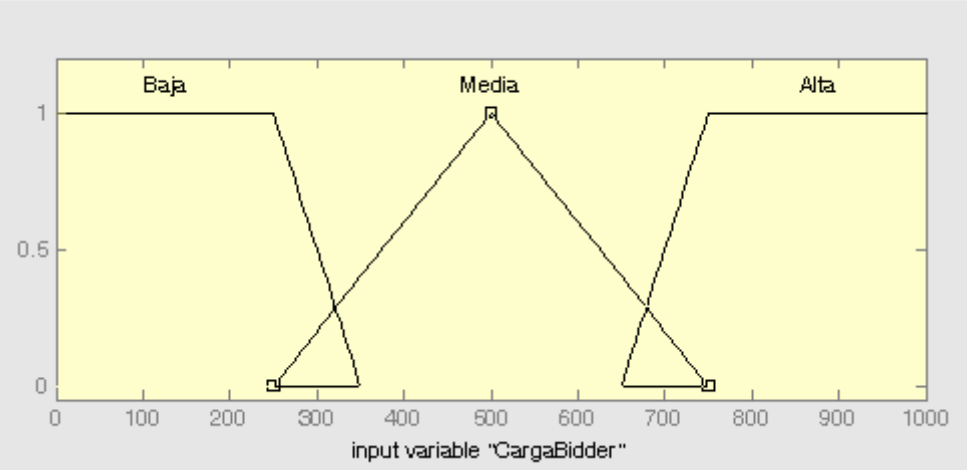


Figure 3 Gráfica de Membresía para CargaBidder.

Tiempo de Invocación del Método Remoto (RMI).

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 la respuesta del 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 [Scallan 00].

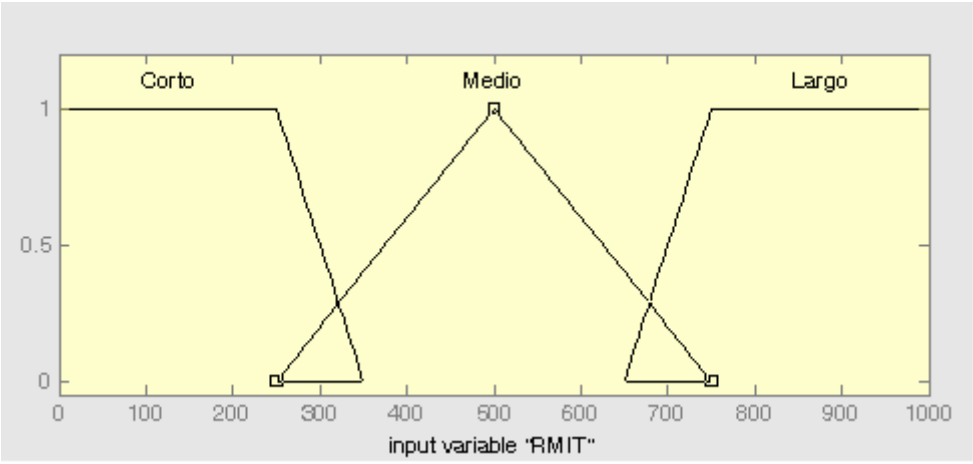


Figure 4 Gráfica de Membresía para RRMIT.

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 esta calculado en milliseconds. El conjunto fuzzy para Reply Remote Method Invocation Time (RMIT) esta definido como: {Corto, Medio, Largo}. La Figura 4 muestra la grafica de membresía para medir el tiempo de invocación del método remoto (RMIT).

Quality Of Service.

Usamos Quality Of Service para clasificar los servicios en seis diferentes categorías: {MuyBaja, Baja, MedioBaja, Media, MedioAlta, Alta}. La gráfica de membresía es mostrada en la Figura 5.

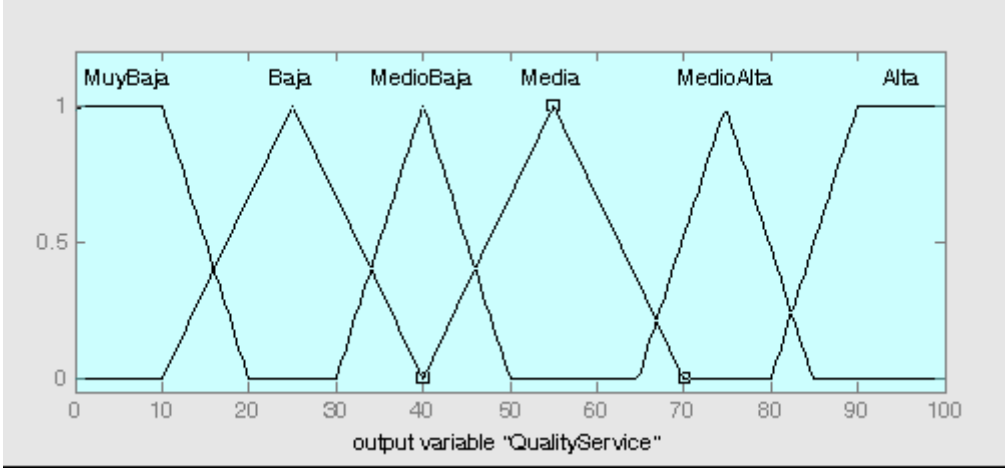


Figure 5 Gráfica de Membresía para Quality Of Service.

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

- 1. **If** (CargaBidder is Baja) **and** (RMIT is Corto) **then** (QualityService is Alta)
- 2. **If** (CargaBidder is Baja) **and** (RMIT is Medio) **then** (QualityService is MedioAlta)
- 3. **If** (CargaBidder is Baja) **and** (RMIT is Largo) **then** (QualityService is Media)
- 4**. If** (CargaBidder is Media) **and** (RMIT is Corto) **then** (QualityService is MedioAlta)
- 5. **If** (CargaBidder is Media) **and** (RMIT is Medio) **then** (QualityService is Media)
- 6. **If** (CargaBidder is Media) **and** (RMIT is Largo) **then** (QualityService is Baja)

- 7. **If** (CargaBidder is Alta) **and** (RMIT is Corto) **then** (QualityService is MedioBaja)
- 8. **If** (CargaBidder is Alta) **and** (RMIT is Medio) **then** (QualityService is Baja)
- 9. **If** (CargaBidder is Alta) **and** (RMIT is Largo) **then** (QualityService is MuyBaja)

Al aplicar las reglas de inferencia, una decisión puede ser generada basada en ambos antecedentes. Esto es, si RMIT es Corto y CargaBidder es Baja, entonces QualityService es Alta (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 RMIT y CargaBidder 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. Usamos el método del centroide para traslapar las áreas [Yen 99, Yu 04].

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

$$\overline{X} = \frac{\sum_{i=1}^{N} \overline{x_i} A_i}{\sum_{i=1}^{N} A_i} \qquad (1)$$

donde A_i y $\overline{x_i}$ son el área traslapada y el centroide de los triángulos o trapecios obtenidos en la i^{th} regla. El centroide y el área son calculados para cada triangulo o trapecio. Este proceso es repetido para otras reglas de inferencia donde las entradas sean aplicadas para obtener un área compuesta de trapecios traslapados. El proceso de defusificación genera un valor centroide que representa el rango de la QoS. El Manager puede entonces enviar los requerimientos del cliente al mejor bidder basándose en QoS obtenida para cada bidder. En la figura 8.19 observamos QualityService cuando tenemos diferentes entradas que representan la carga en los bidders (CargaBidder) y el

tiempo de respuesta que genera cada uno en la red (RMIT).

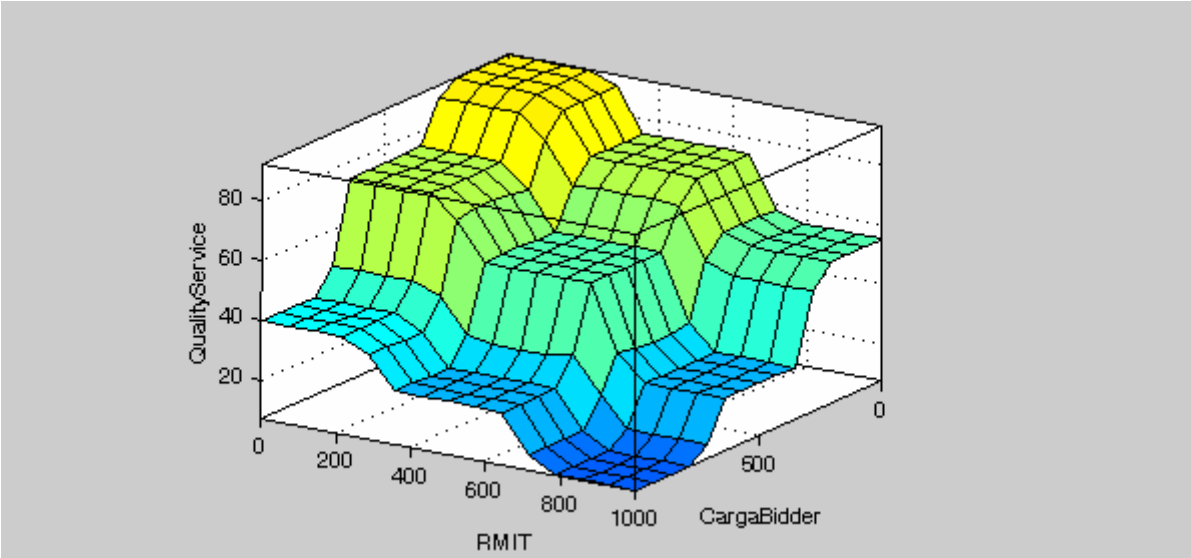


Figure 6 Gráfica de Salida para QualityService.

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.

Número	CargaBidder	RMIT(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 (CargaBidder), Tiempo de Respuesta (RMIT) y Quality of Service (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%). 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 QualityService es medio alta (75%).

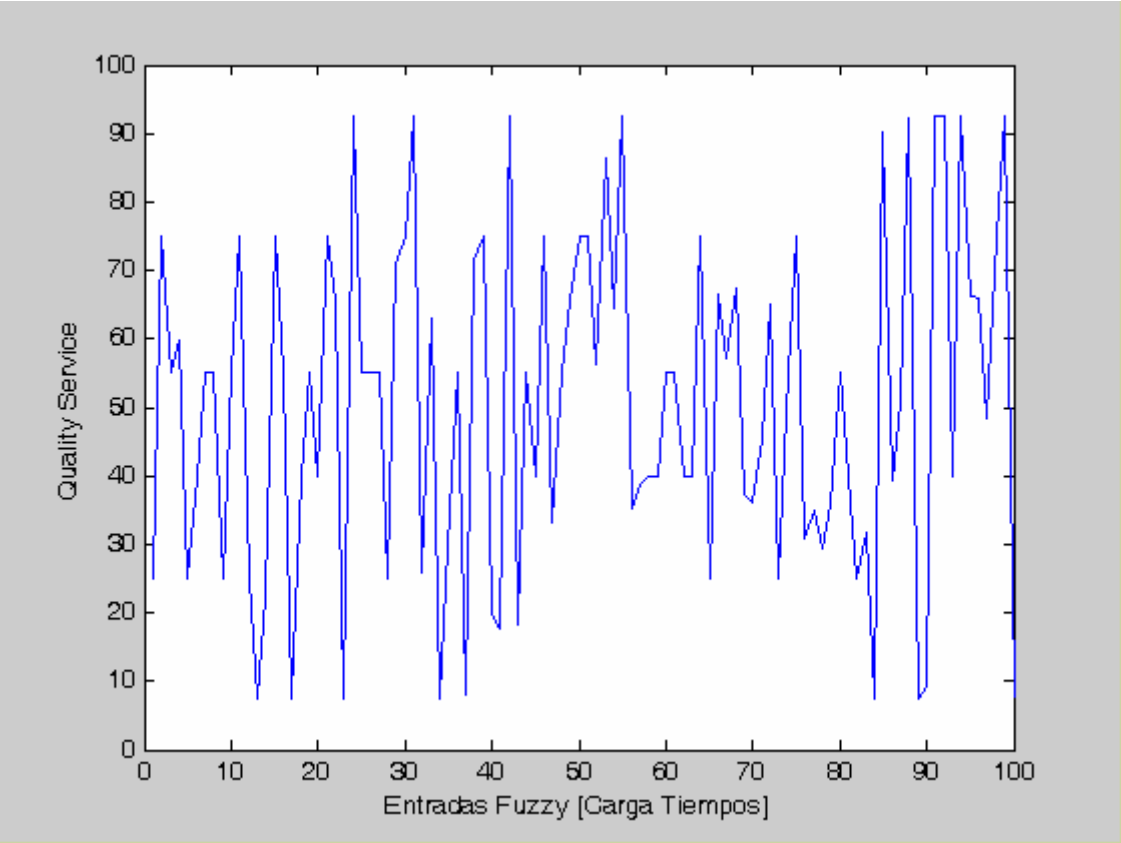


Figura 7 QualityService para 100 Entradas de los Bidders.

En la entrada 13 y 17 cuando los bidders tienen un tiempo de respuesta alto y una carga elevada, nuestro sistema infiere que QualityService es muy mala (8%). La gráfica 7 muestra QualityService, que genera nuestro modelo fuzzy cuando simulamos 100 entradas de los bidders.

Glosario de Términos.

BOA (*Basic Object Adaptor*) : Adaptador básico de objetos.

Bridge : Puente.

Call-back : Llamada en retroceso.

CORBA (*Common Object Request Broker Architecture*) : Arquitectura común para un mediador de

objetos.

DCE (*Distributed Computing Environment*) : Entorno computacional distribuido.

DDCF (*Distributed Document Component Facility*) : Facilidad de distribución de componentes de

documentos.

DII (*Dinamic Invoction Interface*) : Interfaz de invocación dinámica.

Distributed Object Programs : Programas de objetos distribuidos.

DPE (*Distributed Program Environment*) : Entorno de distribución de programas.

DSI (*Dinamic Skeleton Interface*) : Esqueleto de la interfaz dinámica.

ESIOPs (*Environment-Specific Inter-ORB Protocol*) : Protocolo de entorno específico entre ORBs.

GIOP (*General Inter-ORB Protocol*) : Protocolo general entre ORBs.

IDL (*Interface Definition Language*) : Lenguaje de definición de interfaces.

IIOP (*Internet Inter-ORB Protocol*) : Protocolo de internet entre ORBs.

IOR (*Interoperable Object Reference*) : Referencia interoperativa de un objeto.

IR (*Interface Repository*) : Repositorio de interfaces.

NON-PREEMPTIVE : Tarea sin estado de ejecución.

Object Framework : Marco de objetos.

OMA (*Object Management Architecture*) : Arquitectura de administración de objetos o arquitectura

de gestión de objetos.

OMG (*Object Management Group*) : Grupo de gestión de objetos.

ORB (*Object Request Broker*) : Mediador de petición de objetos.

OVERHEAD (Sobrecarga) : Réplica con carga excesiva.

PREEMPTIVE : Tarea con estado de ejecución.

POA (*Portable Object Adaptor*) : Adaptador portable de objetos.

RFPs (*Request For Proposals*) : Petición para propuestas.

Skeleton : Esqueleto.

Stub : Representante (código intermediario).

Throughput (*Portable Object Adaptor*) : Máxima utilización de recursos en periodo de tiempo determinado.