UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

CENTRO UNIVERSITARIO UAEM VALLE DE MÉXICO

BALANCEO-DISTRIBUCIÓN DE CARGA DE TRABAJO Y FRAGMENTACIÓN-REPLICACIÓN DE INFORMACIÓN PARA EL SERVIDOR DE APLICACIONES WEB APACHE TOMCAT

TESIS

Que para obtener el Título de

INGENIERO EN COMPUTACIÓN

P r e s e n t a

C. Victor Hugo Carbajal Rosas

Asesor: Dr. en C. Héctor Rafael Orozco Aguirre

Atizapán de Zaragoza, Edo. de Méx. Noviembre de 2017

Centro Universitario
UAEM Valle de México

Abstract

When it is talked about the term web application, this is understood as that tool that users use to access a web server through internet or a intranet. These applications are accessed in a practical way through web browsers as light clients on the user's side. The ability to update and maintain web applications without distributing and installing software in thousands of potential clients is a reason for its prestige.

In this thesis, a new mechanism of distribution and balancing of work load is proposed for web application server Apache Tomcat 7.0, as well as another one of replication and information fragmentation. For this, a Peer-to-Peer architecture will be implemented in a distributed manner, that will allow the server to automatically and efficiently assign the work to be done among several participating machines, at the same time that it provides them with information they need to be able to carry out the work assigned to them. With that, it seeks to reduce the response time of requests that are received by the server. In this way, it will be possible to attend a greater number of requests, investing thus a smaller amount of time.

Resumen

Cuando se habla del término aplicación web, este se entiende como aquella herramienta que los usuarios utilizan para tener acceso a un servidor web mediante internet o bien una intranet. Estas aplicaciones son accedidas de manera práctica por medio de navegadores web como clientes ligeros del lado del usuario. La capacidad para actualizar y mantener aplicaciones web sin distribuir e instalar software en miles de potenciales clientes es una razón de su prestigio.

En esta tesis, se propone un nuevo mecanismo de distribución y balanceo de carga de trabajo del servidor de aplicaciones web Apache Tomcat 7.0, al igual que otro de replicación y fragmentación de información. Para ello, se implementará una arquitectura Peer-to-Peer de manera distribuida que permitirá que el servidor asigne de manera automática y eficaz el trabajo a realizar entre varias máquinas participantes, al mismo tiempo que les proporciona la información que requieren para poder llevar a cabo el trabajo que les sea asignado. Con ello, se busca reducir el tiempo de respuesta de las solicitudes que sean recibidas por el servidor. De esta manera, será posible el poder atender un mayor número de peticiones, invirtiendo así una menor cantidad de tiempo.

Índice general

Índice de figuras

Capítulo 1

Introducción

Una página web es una fuente de información a la que pueden acceder una gran cantidad de usuarios a la vez mediante cualquier computadora o dispositivo móvil que cuente con acceso a la World Wide Web [1] a través de un navegador web. En una página web, la información es presentada en formato HTML [2], o bien, en otros más recientes como JSP [3], JSF [4] ASP [5] o PHP [6].

1.1. Antecedentes

Desde una página web, se puede tener acceso mediante ligas o enlaces a otras más que pueden ser dinámicas o estáticas y que se encuentran en la World Wide Web. Estas pueden ser cargadas desde una computadora local o desde una remota conocida como servidor web. Es este último, quien denega accesos a las mismas a una red privada, un claro ejemplo es una intranet [7], o bien, puede dar un libre acceso en el World Wide Web por medio del Internet [8]. Las páginas web son solicitadas y transferidas de los servidores usando el Protocolo de Transferencia de Hypertexto [9] (HTTP, por sus siglas en Inglés Hypertext Transfer Protocol).

Por lo general, el tipo de información que se publica en una página web puede provenir de archivos de texto estático y gráficos fijos, o se puede generar como una serie de archivos con código generado al momento, o bien predeterminado, que le indique al servidor cómo construir el HTML para cada página web solicitada, a esto se le conoce como página web dinámica [10].

Cuando se habla del término aplicación web, este se entiende como aquella herramienta que los usuarios utilizan para tener acceso a un servidor web mediante Internet o bien una intranet. Estas aplicaciones son accedidas de manera práctica por medio de navegadores web como clientes ligeros del lado del usuario. La capacidad para actualizar y mantener aplica-

ciones web sin distribuir e instalar software en miles de potenciales clientes es una razón de su prestigio. Una gran variedad de aplicaciones como lo son el correo web, wikis, weblogs, tiendas en línea, entre otras, son ejemplos bien conocidos de estas. En las aplicaciones web, se crean dinámicamente una variedad de páginas en formato estándar, HTML o XHTML, tolerado por los navegadores web más comunes. Para ello, se emplean lenguajes interpretados del lado del cliente, tales como JavaScript [11] o VBScript [12], para añadir elementos dinámicos a la interfaz de usuario.

Básicamente, cada página web de manera individual es enviada al usuario como un documento estático, pero el seguimiento de las páginas da una sensación de una experiencia interactiva. Una aplicación web está comúnmente estructurada como una aplicación de tres capas:

- Capa 1: en su aspecto más básico, es el navegador web.
- Capa 2: cualquier tecnología web dinámica (CGI [13], PHP, JSP o ASP).
- Capa 3: el mantenimiento de una base de datos.

En la figura 1.1, se observa que dichas capas funcionan todas en conjunto cuando el navegador web envía peticiones al servidor y este las realiza valiéndose de altas, bajas, consultas y actualizaciones a la base de datos y responde notificando los resultados obtenidos, los cuales son mostrados en la interfaz de usuario en el navegador web. Aquí surge un gran problema, este es cuando al mismo tiempo demasiadas peticiones son enviadas por los usuarios, lo que ocasiona que el servidor debido a su política de atención de tipo primero en entrar, primero en salir (FIFO, por sus siglas en Inglés First In, First Out) se vuelva un cuello de botella. Lo anterior, se presenta cuando la cantidad de peticiones sobrepasa el límite de atención del servidor para responder de manera oportuna. Con ello, se pueden tener problemas derivados que recaigan en casos en donde nunca se atiendan peticiones ya que el servidor puede colapsar cuando se sobrepasa su número máximo tolerable a ser atendido.

1.2. Planteamiento del problema

En la actualidad, el uso de aplicaciones o páginas web se ha vuelto muy popular en el mundo. No obstante, debido a la gran cantidad de recursos que deben ser gestionados, una sola computadora no puede controlar el manejo de estos, por esta razón se implementan servidores de aplicaciones web. A pesar de ello, un único servidor en la mayor parte de las veces no es suficiente debido a la gran cantidad de usuarios que acceden a las páginas que se administran. Este factor se ve reflejado en lo que se conoce como cuello de botella, el

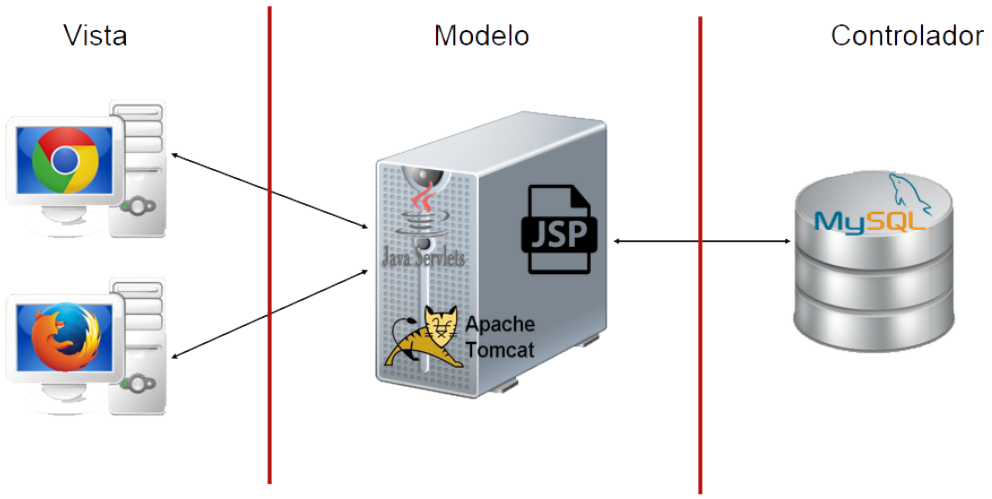

Figura 1.1: Capas de aplicaciones web

cual para poder evitarlo implica que se recurra a la ayuda de otros servidores auxiliares o máquinas secundarias para agilizar el trabajo de atención de peticiones y envío de respuestas.

La razón principal que origina el problema antes mencionado es debido a que los servidores de aplicación web atienden solicitudes encolándolas, esto quiere decir que el orden en el cual son recibidas será el mismo en que estas son atendidas. Al utilizar una cola de peticiones, se ven más desventajas que ventajas debido a que los tiempos de espera para algunos usuarios pueden ser considerablemente grandes, aunque muy pocas veces no lo es así. Para explicar esto, si el número de usuarios se ve incrementado es lógico pensar que el servidor se vuelve un cuello de botella, porque no es capaz de atender demasiadas solicitudes a la vez, esto provocará que algunas de ellas si sean atendidas en un corto plazo, pero el resto que sería la mayoría lo sea a mediano o largo plazo, o bien, en el peor de los casos que no puedan ser atendidas. Existen demasiadas razones que pueden explicar este fenómeno, pero una muy clara sería cuando el servidor pudo haber quedado fuera de funcionamiento antes de que la solicitud fuera enviada, después o justo al momento de recibirla, o bien, poco tiempo antes de enviar la respuesta de atención de la misma.

Si se desea que los servicios mejoren en cuestión a su velocidad de operación y se eviten al máximo los cuellos de botella, se deben de plantear nuevos esquemas de trabajo para los mismos, es donde la teoría de los sistemas distribuidos [14] da la clave que se necesita para poder lograrlo al optar por una buena solución en el ámbito de las aplicaciones web.

Para atacar este problema, se pueden emplear técnicas de balanceo y distribución de carga de trabajo sin olvidar que con la información se debe recurrir a técnicas de fragmentación y replicación de la misma. Para demostrar que es posible darle solución a dicho problema, en

este trabajo de tesis se plantea una propuesta basada en la teoría de los sistemas distribuidos para el caso del servidor Apache Tomcat 7.0 [15]. Con ello, demostrará que dicho servidor tendrá un mejor rendimiento para la atención de múltiples solicitudes de varios usuarios a la vez.

1.3. Justificación

Los servidores de aplicaciones web a menudo se vuelven un cuello de botella cuando la cantidad de peticiones que están en espera de ser atendidas supera al número máximo posible a atender por unidad de tiempo. Al recurrir a las técnicas de distribución y balanceo de carga de trabajo así como a las de fragmentación y replicación de información, se pueden usar más máquinas participantes como servidores secundarios para disminuir los tiempos de respuesta y atender con ello a una mayor cantidad de solicitudes en un tiempo considerablemente mucho menor y mejor administrado. Junto con estas técnicas se debe emplear una arquitectura de red entre pares (P2P, por sus siglas en Inglés Peer-to-Peer) [16], para que cada máquina funja a conveniencia entre ser cliente y ser servidor sin que se pierda la armonía del sistema.

Una arquitectura P2P hace referencia a un tipo de estructura de organización entre computadoras que permite la comunicación e interacción entre varios usuarios de manera bidireccional, lo que les permite una interacción mucho más fluida entre sí y más posibilidades de compartir recursos, con la ventaja de que no es necesario el uso de un servidor central que facilite o administre dicha comunicación. Aquí, se tiene la gran ventaja de contar con una mayor confiabilidad y robustez en los procesos de envío-recepción de mensajes entre máquinas.

1.4. Motivación

Dentro de las habilidades adquiridas a lo largo de su estadía en una institución, un Ingeniero en Computación debe ser capaz de ofrecer soluciones a diversos problemas, por citar un problema claro, cuando en una institución de educación superior se está llevando a cabo el proceso de re inscripción a un nuevo ciclo escolar, se experimentan situaciones tales como:

- El servidor web al tratar de procesar varias peticiones a la vez, este se haga lento.
- Colapso del servidor por exceso de peticiones.
- Al recibir varias peticiones a la vez, genere la perdida de las peticiones o respuestas.

- Mayor número de máquinas con acceso a la información replicada.
- En caso de contingencia se tiene un backup actualizado en otra parte
- Entre más máquinas participantes existan, mayor serán el número de réplicas de información.

A partir de lo anterior, el autor de este trabajo decidió ofrecer una solución para este tipo de problemas que existen en los servidores de aplicaciones web, ya que estos en la actualidad son muy frecuentes, debido a que es común encontrar sistemas que requieran de servidores para su funcionamiento, tales como el Apache Tomcat 7.0, y en ocasiones su uso es referente a tareas muy importantes que necesitan ser procesadas de tal manera que no exista algún tipo de interrupción a la hora de ser atendidas, los tiempos de atención a las peticiones pueden ser muy largos cuando el número de peticiones en espera excede el límite del servidor.

Por este tipo de razones, la implementación de nuevos mecanismos que realicen el balanceo y distribución de carga de trabajo, así como la fragmentación y replicación de información en servidores web, recaen en un nuevo enfoque que propone el autor de este trabajo para dar solución a los problemas mencionados anteriormente.

1.5. Hipótesis

En esta tesis, se dará cumplimiento a los siguientes enunciados:

- Al tener una o más máquinas participantes los tiempos de respuesta deben reducirse de manera proporcional a la cantidad de ellas.
- El balanceo y distribución será tan equitativo como sea posible para evitar que una máquina se sobre sature, o bien, que dure el menor tiempo posible en espera de solicitudes a atender (que no esté ociosa).
- Una gran ventaja que tiene el mecanismo será derivado de la comunicación P2P, así cuando una máquina tenga problemas de funcionamiento esta no afectará la estabilidad del sistema ya que otra podrá suplirla para afectar lo menos posible los tiempos de respuesta. Además, toda máquina puede fungir como cliente o servidor, o ambos, según sean las necesidades del sistema.
- La información debe estar al alcance de las máquinas participantes para que sea accedida al momento sin que se afecten los tiempos de respuesta.

- La fragmentación y replicación de información debe ser hecha de manera constante para que siempre este actualizada y sean atendidas de manera confiable las solicitudes recibidas. Cabe señalar, que según se incremente el número de máquinas participantes y se tenga un mayor número de peticiones la hipótesis antes citada no es sencilla de satisfacer. Sin embargo, en este trabajo se ofrece la certeza de que sí será cubierta de manera satisfactoria.

1.6. Objetivos

Los objetivos de este trabajo de investigación son los siguientes:

1.6.1. Objetivo general

Implementar una arquitectura P2P de manera distribuida que permitirá que una máquina con el servidor Apache Tomcat 7.0 distribuya y balancee de manera automática y eficaz a través de un proceso monitor el trabajo (peticiones) a realizar a máquinas participantes que funjan como servidores secundarios, las cuales mediante fragmentación y replicación de información tendrán los datos necesarios para llevar a cabo su trabajo.

1.6.2. Objetivos específicos

1. Disminuir los tiempos de respuesta en la realización de peticiones para atender la mayor cantidad de usuarios y peticiones en un mismo intervalo de tiempo.
2. Distribuir lo más equitativo posible la cantidad de trabajo entre las máquinas participantes por medio del proceso monitor.
3. Asignar a cada máquina participante las sentencias de creación de la base de datos para que manejen su respectiva información, logrando tener un mecanismo de fragmentación.
4. Enviar desde cada máquina participante al proceso monitor los fragmentos de información contenidos en su respectiva base de datos, cumpliendo con el mecanismo de replicación.
5. Implementar el uso del protocolo TCP [17] para evitar la pérdida de paquetes durante el envío de estos.
6. Crear una aplicación web para reportes y consultas de incidentes de tráfico vehicular, misma que sea empleada para validar los mecanismos de distribución-balanceo de carga de trabajo y fragmentación-replicación de información.

1.7. Delimitación y alcance

La propuesta planteada en esta tesis será para el servidor Apache Tomcat 7.0. Para demostrar que se cumplirán los objetivos planteados, el alcance de este proyecto llegará a la implementación de casos de estudio simples para atender peticiones de altas, bajas y consultas de datos de alumnos, profesores, carreras y unidades de aprendizaje en un contexto similar al actual del Centro Universitario UAEM Valle de México de la Universidad Autónoma del Estado de México. Se utilizará la tecnología J2EE [18] para la implementación de la propuesta.

1.8. Estructura de la tesis

El restante contenido de este trabajo de tesis va a ser abordado conforme a la siguiente organización:

- En el siguiente capítulo, se dará a conocer el concepto de sistemas distribuidos, su forma de implementación y se explicará el modelo cliente-servidor [19], las llamadas a procedimientos remotos y la comunicación en grupo que será implementada en este tema de investigación.
- La revisión y análisis del estado del arte y trabajo relacionado más relevante de las técnicas de distribución y balanceo de carga de trabajo será cubierto dentro del tercer capítulo.
- En el cuarto capítulo, se explicará lo referente a la replicación de información, sus técnicas y tipos que existen.
- En el capítulo cinco, se dará a conocer lo estudiado sobre la fragmentación de información, los grados y reglas que se tienen que tomar en cuenta antes de implementarla.
- En el penúltimo capítulo, se expondrá la propuesta planteada en este trabajo de tesis, así como la aplicación web y caso de estudio realizados.
- Finalmente, las conclusiones y trabajo a futuro serán dados en el último capítulo.

Capítulo 2

Sistemas distribuidos

Cuando una tarea está siendo llevada a cabo la confiabilidad de un sistema distribuido debe ser una de las principales prioridades, ya que si uno de sus componentes falla, este debe ser reemplazado, o bien, buscar que sea corregido todo problema que el mismo presente para garantizar que la información no se pierda o bien se corrompa. No obstante, cuando el número de componentes es muy grande, no siempre resulta trivial todo aquel mecanismo que deba ser puesto en marcha, ya que el número de componentes puede ir desde unos cuantos hasta miles o más de cientos de miles. En este capítulo se darán a conocer todas las características de un sistema distribuido y los conceptos más relevantes que giran en torno a ellos.

2.1. ¿Qué es un sistema distribuido?

Aunque existen varias definiciones en la literatura, a continuación se dan dos de las más claras [20]:

- Colección de computadoras o máquinas conectadas por una red de comunicaciones, en la cual el usuario percibe la apariencia de interactuar con una sola computadora, debido a que el mismo puede acceder a los recursos remotos de la misma manera en que accede a los recursos locales.
- Serie de dispositivos localizados en una red de computadoras que se comunican y coordinan para llevar a cabo sus tareas mediante el paso de mensajes.

Analizando las anteriores definiciones, surgen los siguientes aspectos a ser tomados en cuenta: concurrencia de los componentes, falta de un reloj global y fallas independientes en los componentes. Además, deben contemplarse las siguientes características inherentes de los mismos:

- Sistemas autónomos capaces de comunicarse y cooperar mediante interconexiones hardware y software, para proporcionar una abstracción de máquina virtual a los usuarios.
- Un objetivo clave es la transparencia, la cual se logra al compartir de manera global los recursos para asemejar que existen a nivel local.

2.1.1. Metas de los sistemas distribuidos

- La primera de ellas y que nunca debe fallar ni faltar es mantener siempre una conexión entre los usuarios y los recursos que necesitan.
- Tener transparencia de distribución para que no sepa por parte de usuario que computadoras llevaron a cabo las tareas que este solicita.
- Deben ser abiertos hasta un cierto grado, para que nuevos servicios de compartición de recursos tengan la capacidad de ser añadidos, siempre y cuando no perjudiquen ni dupliquen a los ya existentes.
- Tener escalabilidad para futuras optimizaciones tanto de software como de hardware para brindar servicio al mayor número de usuarios en el menor tiempo posible.

Existen 3 diferentes formas de procesamiento distribuido:

- Basado en llamadas a procedimientos remotos: invocar procesos remotos como si fuera de manera local. Aquí, los detalles de envío-recepción de mensajes quedan ocultos para el usuario y se cumple con la transparencia.
- Basado en objetos distribuidos: cuando la invocación a los procesos remotos es similar a invocar métodos de objetos. Por ende, se tiene un enfoque de nivel más alto al anterior y se cumple por igual con la transparencia.
- Basado en memoria compartida: ocurre en casos en los cuales un proceso necesita leer o escribir datos en una memoria común para diferentes computadoras. En este sentido, se necesita contar con un mecanismo fiable de sincronización de procesos.

2.2. Modelo cliente-servidor

Este modelo visto como una arquitectura de comunicación en red que permite al usuario en una máquina, llamada el cliente, requerir algún tipo de servicio de una máquina a la que está unido, llamado el servidor, mediante una red de área local (LAN, de su nombre en

inglés Local Area Network) o una red de área amplia o ancha (WAN, de su nombre en inglés Wide Area Network). Estos servicios pueden ser peticiones de datos de una base de datos, de información contenida en archivos o los archivos en sí mismos, entre más como peticiones de imprimir datos en una impresora asociada.

Aunque clientes y servidores suelen verse como máquinas separadas, pueden, de hecho, ser dos áreas o procesos separados en la misma máquina. Por lo tanto, una única máquina puede ser al mismo tiempo cliente y servidor. Además, una máquina cliente unida a un servidor puede ser a su vez servidor de otro cliente y el servidor puede ser un cliente de otro servidor en la red tal como se muestra en la figura 2.1.

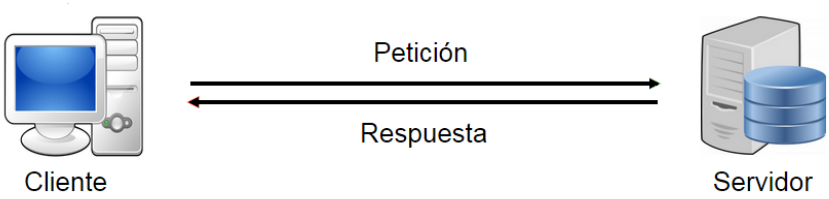

Figura 2.1: Funcionamiento del modelo cliente-servidor

2.2.1. Máquina o proceso cliente

Es quien empaqueta los requerimientos que el usuario formuló en su petición para que sean enviados al servidor. Esto dice que el cliente es el encargado del manejo, manipulación y despliegue de los datos para con el usuario, dando paso a que permita una interacción con el usuario por medio de interfaces gráficas para tener acceso a los servicios distribuidos que se encuentren en cualquier componente o nodo de la red.

Las tareas principales que realiza el cliente son las siguientes: administrar la interfaz de usuario, mantener una interacción confiable con el usuario, procesar la lógica de la aplicación y hacer validaciones locales, generar requerimientos de bases de datos, recibir los resultados que fueron generados por el servidor y formatear dichos resultados.

2.2.2. Máquina o proceso servidor

Es el opuesto al cliente, ya que es visto como el encargado de atender peticiones de manera secuencial. Es decir, el servidor normalmente maneja todas las funciones relacionadas con la mayoría de las reglas del negocio y los recursos de datos.

Las funciones que lleva a cabo el proceso servidor se resumen como sigue: aceptar los

requerimientos de bases de datos que hacen los clientes, procesar requerimientos de bases de datos, formatear datos para trasmitirlos a los clientes, procesar la lógica de la aplicación y realizar validaciones a nivel de bases de datos.

2.3. Llamadas a procedimientos remotos

Las llamadas a procedimientos remotos (RPC, de su nombre en inglés Remote Procedure Calls) [21] es un modelo de programación de alto nivel que abstrae y oculta los detalles de bajo nivel de comunicación, al mismo tiempo que evita tener que tratar la problemática de la portabilidad y dependencias de hardware. De este modo, RPC permite a los programadores expresar la comunicación entre el cliente y el servidor como si fuera una llamada a un procedimiento local. Este es empleado para permitir el desarrollo de sistemas de procesamiento distribuido basados en el paradigma procedimental.

Cabe mencionar, que a nivel local cuando se realiza una llamada a un procedimiento, función o subrutina, se hace con el fin de transferir el control de una parte del programa a otra. A nivel remoto con el uso de RPC se logra un efecto similar y transparente para el usuario como se muestra en la figura 2.2. En dicha figura, se encuentran asociados el paso de parámetros y el retorno de uno o varios resultados al realizar una RPC.

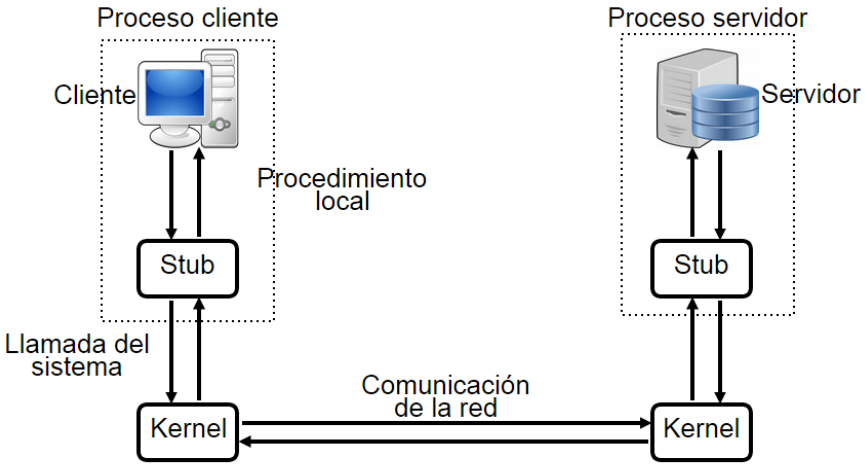

Figura 2.2: Funcionamiento de RPC

2.3.1. Componentes de un sistema de RPC:

1. Tiempo de construcción:

- Compilador del Lenguaje de Definición de Interfaz (IDL, Interface Definition Languaje): es un lenguaje que permite interactuar a los objetos de una forma independiente del lenguaje, es decir, que no importa el lenguaje con el que estos fueron escritos.
- Define la interfaz de servicio (resguardos): integra el mecanismo RPC con los programas del cliente y el servidor estando estos escritos en lenguajes de programación convencional.
- Mantiene una heterogeneidad: hace que este se mantenga disponible para múltiples lenguajes de programación.

2. Tiempo de ejecución:

   - Todos los parámetros del procedimiento objetivo junto con sus argumentos son empaquetados en un mensaje y le solicita al módulo de comunicación local que lo envié al resguardo servidor.
   - Los módulos de resguardos son incluidos en el cliente servidor.
   - Oculta la comunicación y crea una abstracción de llamada a procedimiento.
   - Cuando recibe los resultados de la ejecución del proceso, los desempaqueta y los envía al cliente.

3. Resguardos del cliente (stubs cliente):

   - Este inicializa la RPC y realiza una llamada la cual invoca al resguardo.
   - Es una capa que proporciona los servicios que dan soporte a RPC (por ejemplo un ligado de datos, una conversión de datos, la autenticación, el tipo de comportamiento que se tomara ante algún tipo de falla) a través de la red entre las máquinas participantes como cliente-servidor.

4. Resguardos del servidor (stub servidor):

   - Este trabaja de una forma simétrica tal y como lo hace el resguardo cliente.
   - Cada vez que este recibe un requerimiento de llamada del resguardo servidor, este inicializa el procedimiento apropiado y regresa el resultado desde el mismo resguardo servidor.
   - El ligador toma los parámetros del mensaje de petición, los coloca en la pila, localiza los servicios requeridos y las alternativas en dos ámbitos diferentes:
   - En ámbito no global, hace uso del identificador de servicio más la dirección del servicio.

- En ámbito global, sólo hace uso del identificador de servicio.
- Por último pone los parámetros de resultado en un mensaje de respuesta y por medio de primitivas de comunicación envía el mensaje al resguardo cliente.

2.3.2. Transparencia de una RPC

Para garantizar que el nivel de transparencia en una RPC es aceptable, se debe contemplar lo siguiente:

- Transparencia sintáctica: se tiene cuando una llamada a procedimiento remoto tiene el mismo tipo de sintaxis que una llamada local.
- Paso de parámetros: estos son pasados por valor, en cada procedimiento remoto son definidos los argumentos de entrada y los valores de retorno.
- Enlace: el cliente es el que llama al sistema remoto apropiado para la ejecución de un procedimiento en específico.
- Protocolo de transporte de datos: solamente soporta uno o dos protocolos, en específico las implementaciones ONC y DDE soportan TCP y UDP como protocolos de transporte.
- Manejo de excepciones: sobre un ambiente de sistemas distribuidos las posibilidades de falla aumentan, es por eso que se deben proveer mecanismos para controlar estas situaciones.
- Transparencia semántica: es la misma que se usa para una llamada local, existen 3 tipos de semántica en RPC las cuales son: exactamente una vez, cuando mucho una vez y al menos una vez.
- Representación de los datos: todas las implementaciones definen uno o más tipos de datos estandar para los tipos de datos soportados por la implementación.
- Desempeño: el desempeño en una RPC disminuye entre 10 y 100 veces en comparativa a una llamada a un procedimiento local.
- Seguridad: con RPC existen los mismos problemas de seguridad que los relacionados con la ejecución remota de comandos.

2.3.3. Fundamentos de RPC

Los principales fundamentos que debe cumplir RPC son los siguientes:

1. Código añadido a servicios en sistemas distribuidos:
   - No depende de la implementación cliente-servidor.
   - Solamente es dependiente de la interfaz de servicio.
   - Puede ser generado de forma automática a partir de la interfaz de servicio.
2. Objetivos de RPC:
   - Proveer de servicios al igual que en los sistemas no distribuidos.
   - Sólo hay que programar las bibliotecas de servicios y las aplicaciones.
   - El código restante se genera automáticamente.
   - Lograr una semántica convencional de llamadas a procedimientos remotos en los sistemas distribuidos.
3. Implementaciones más populares de RPC:
   - La que fue desarrollada por Sun Microsystems denominada ONC-RCP (Open Network Computing), esta fue distribuida por todos los sistemas UNIX.
   - La que fue desarrollada por Microsoft en línea con el Ambiente de Computación Distribuida (DCE, Distributed Computing Enviroment) definido por la Fundación de Software Abierto (OSF, Open Software Foundation), así que es incluida en los sistemas operativos Windows.

Cuando el código que invoca a un procedimiento y dicho procedimiento está en un mismo proceso en una computadora dada, se dice que ha ocurrido una llamada a un procedimiento local. Por el contrario, en una llamada a un procedimiento remoto el sistema local invoca, a través de la red, a una función alojada en otro sistema. Lo que se pretende es hacerle parecer al programador que está ocurriendo una simple llamada local.

2.4. Comunicación en grupo

Cuando se habla de una RPC se habla de que la comunicación se lleva a cabo entre dos procesos, el cliente y el servidor, muchas de las veces la comunicación no solo puede ocurrir entre dos procesos, sino que también pueden ser realizados entre varios procesos.

En la comunicación en grupo un mensaje puede ser enviado a varios receptores sin necesidad de realizar múltiples operaciones, la idea es ocultar el envío de los mensajes haciendo que el aspecto de la comunicación luzca como una llamada a procedimiento ordinario.

Hay cuatro tipos de grupos de comunicación, los cuales se explican a continuación:

2.4.1. Uno a uno

Uno a uno o también llamado unidifusión (unicast, por su nombre en Inglés) es el término usado para describir a la comunicación en donde una parte de información es enviada de un punto a otro haciendo referencia de los sistemas distribuidos que solo son un emisor y un receptor (ver figura 2.3).

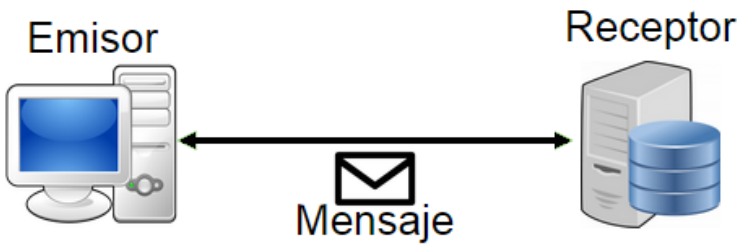

Figura 2.3: Comunicación en grupo, uno a uno

2.4.2. Uno a muchos

En la comunicación uno a muchos donde una parte de información es enviada de uno o más puntos hacia un conjunto de puntos. En este caso aquí hay uno o más emisores y la información es distribuida en un conjunto de receptores (ver figura 2.4).

Aquí los procesos receptores de los mensajes constituyen un grupo, que a su vez pueden ser de tres tipos:

- Grupos cerrados: sólo los miembros del grupo pueden enviar mensajes al grupo. Un miembro externo solo puede enviar mensajes a un proceso individual y no al grupo.
- Grupos abiertos: cualquier proceso en el sistema puede enviar un mensaje al grupo como tal. Un sistema de pasaje de mensajes con la facilidad de grupo de comunicación provee la flexibilidad de crear y borrar grupos dinámicamente y permitir a un proceso agregarse o dejar un grupo.

- Grupo mixto: cuando se tienen operaciones internas y externas en los grupos. Un mecanismo para realizar todo esto es un servidor de grupos. Esta solución sufre de pobre confiabilidad y pobre escalabilidad.

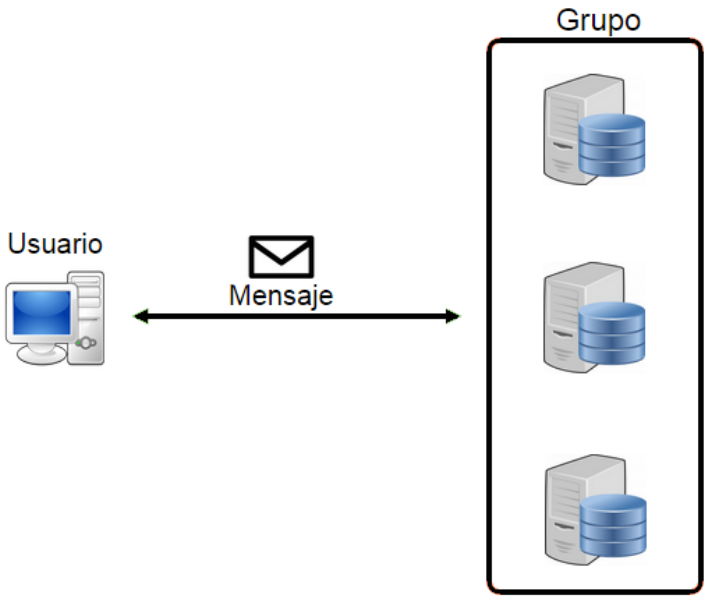

Figura 2.4: Comunicación en grupo, uno a muchos

2.4.3. Muchos a uno

En este tipo de comunicación no existe gran diferencia en cuanto a la comunicación uno a muchos, más que aquí hay n número de emisores los cuales le están enviando mensajes a un único receptor, aquí existe un no determinismo como lo muestra la figura 2.5.

2.4.4. Muchos a muchos

En esta comunicación múltiples emisores envían mensajes a múltiples receptores, no importa que emisor sea o de donde provenga ya que cualquier emisor se puede unir al grupo pudiendo así ver lo que se está enviando como se muestra en la figura 2.6.

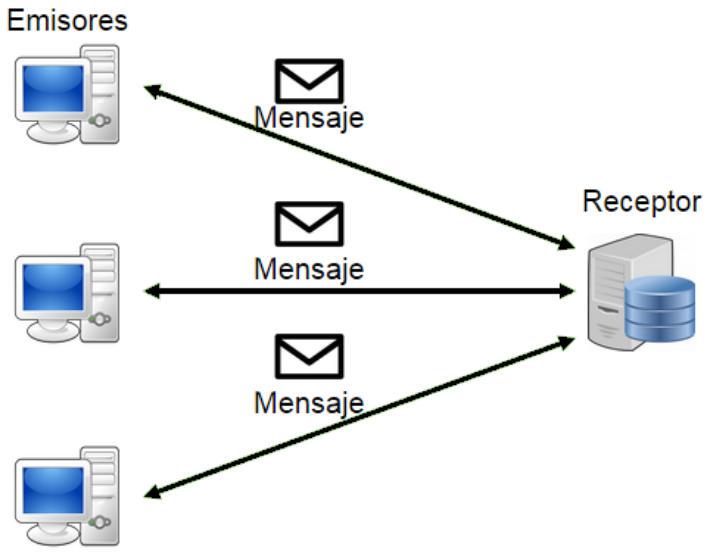

Figura 2.5: Comunicación en grupo, muchos a uno

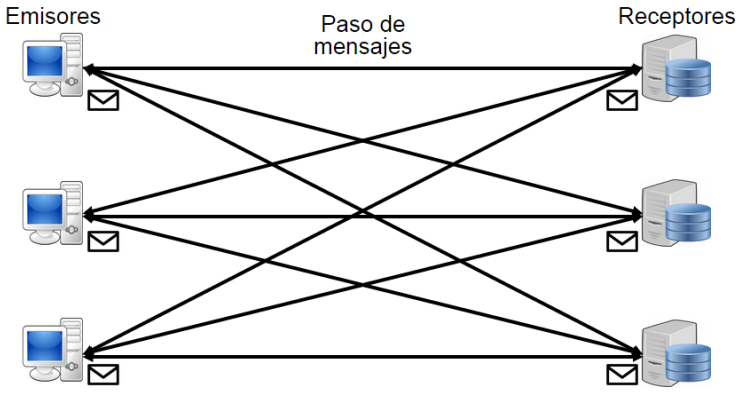

Figura 2.6: Comunicación en grupo, muchos a muchos

2.5. Sincronización entre procesos

Un sistema operativo multiprogramado es un caso particular de sistema concurrente donde los procesos compiten por el acceso a los recursos compartidos o cooperan dentro de una misma aplicación para comunicar información. Ambas situaciones son tratadas por el sistema operativo mediante mecanismos de sincronización que permiten el acceso exclusivo de forma coordinada a los recursos y a los elementos de comunicación compartidos.

Según el modelo de sistema operativo descrito anteriormente, basado en colas de procesos y transiciones de estados, los procesos abandonan la CPU para pasar a estado bloqueado cuando requieren el acceso a algún dispositivo, generalmente en una operación de E/S, pasando a estado preparado cuando la operación ha concluido y eventualmente volver a ejecución. La gestión de estos cambios de estado, es decir, los cambios de contexto, es un ejemplo de sección crítica de código dentro del sistema operativo que debe ser ejecutada por éste en exclusión mutua. Otros ejemplos de código que debe protegerse como sección crítica incluyen la programación de los dispositivos de E/S y el acceso a estructuras de datos y buffers compartidos.

Capítulo 3

Balanceo y distribución de carga de trabajo

Un estudio sobre el balanceo de carga de trabajo es sumamente importante, esto para obtener una distribución de forma equilibrada sobre la carga de procesamiento que será repartida entre todas las máquinas o servidores participantes, con ello se lograra obtener la máxima velocidad de ejecución posible y reducir los tiempos de respuesta.

El procesamiento distribuido permite mejorar el uso de equipos y mejorar el balanceo del procesamiento dentro de una aplicación, este llega a tener un gran impacto ya que en algunas aplicaciones simplemente no existe una máquina que pueda ser capaz de realizar todo el procesamiento.

3.1. ¿Qué es el balanceo de carga de trabajo?

Desde el enfoque del procesamiento distribuido, se manejan varios procesos, cada uno de ellos ejecutándose de manera simultánea en la misma computadora o distribuidos entre varias computadoras conectadas a través de una red, estas colaboran para la realización de una tarea que puede ser tan sencilla como distribuir la carga de trabajo entre varios procesos idénticos por ejemplo el caso de una red de cajeros automáticos, o tan compleja como una multitud de procesos totalmente interdependientes.

Dado lo anterior, el balanceo de carga de trabajo se puede definir como una técnica que incrementa los recursos, explotando el paralelismo y disminuyendo el tiempo de respuesta mediante la distribución de carga apropiada, o visto desde otro punto, es la división de la carga de trabajo de una computadora o servidor central que se le ha asignado, para que este sea realizado entre dos o más computadoras conectadas al servidor central, este tipo de división de carga permite realizar la petición o tarea en un intervalo de tiempo mucho más

reducida, y así lograr realizar más carga de trabajo en el mismo intervalo de tiempo total.

Hay una variedad de formas de realizar el balanceo de carga [22], ya que pueden ser por hardware mediante un sistema de nombres de domino (DNS, por sus siglas en Inglés, Domain Name System) o software, o bien, una combinación de ambas. El balanceo de carga está dirigido primordialmente para entornos que tienden a tener dificultades al procesar todas las peticiones, por su volumen de carga de trabajo.

Puede definirse a la división de carga de trabajo, el proporcionar mayor o menor carga a cada uno de las máquinas participantes, a esta característica se le llama carga asimétrica.

3.2. Técnicas de balanceo de carga de trabajo

Existen una gran variedad de técnicas para llevar a cabo el balanceo de carga de trabajo [23], una técnica es un conjunto de procedimientos para llevar a cabo una determinada actividad, entonces una técnica para el balanceo es un conjunto de procedimientos para repartir el trabajo entre los diversos procesadores participantes dentro de una red, estos tendrán los recursos necesarios para llevar a cabo cada una de las tareas que les sean asignadas individualmente, el balanceo se lleva a cabo en base a diversos algoritmos que dividen lo más preciso y equitativamente posible el trabajo a realizar.

3.2.1. Balanceo estático

De igual manera, conocido como mapeo del problema o planificación del problema, este tipo de balanceo de carga se verifica antes de que se inicialice la ejecución de un proceso cualquiera. El balanceo de carga estático mantiene serias desventajas sobre su equivalente balanceo de carga dinámico, a continuación se hace mención de estas desventajas:

- Es de gran dificultad poder conocer el tiempo total que tardaran en ser ejecutadas todas las partes en que es dividida una petición sin ser ejecutadas.
- En algunas ocasiones las peticiones requieren un número indeterminado de procesamiento computacional para lograr obtener la solución más óptima.
- No todos los sistemas están exentos de algún tipo de retardo, ya que algunos pueden tener pequeñas variantes de retardos con las comunicaciones por diferentes circunstancias, lo que provoca dificultad al incorporar la variable retardo de comunicación sobre el balance de carga estático.

3.2.2. Balanceo dinámico

Este es muy diferente al balanceo estático ya que todos los conflictos que presenta el balanceo de carga estático ya se tienen contemplados, esto se ha logrado porque al dividir la carga computacional total se da prioridad a las tareas que se están ejecutando y no a la estimación del tiempo que pueden tardar en ejecutarse, pero el balanceo de carga dinámico genera consigo una cierta cantidad de sobrecarga durante la ejecución del algún programa, esto resulta como una alternativa mucho más fiable que el balanceo de carga estático.

Aquí las tareas son repartidas entre todos los servidores participantes durante la ejecución de una solicitud, dependiendo de cómo y dónde se almacenen y además de como sean repartidas las tareas entre los servidores, el balanceo de carga dinámico se divide en:

- Balanceo de carga dinámico distribuido: hace uso de n número de nodos maestros y cada uno controla a un grupo de N nodos esclavos.
- Balanceo de carga dinámico centralizado: va de la mano con la estructura conocida como maestro-esclavo.

3.2.2.1. Balanceo centralizado

Aquí el nodo maestro es el responsable de toda la información del total de tareas que se van a realizar, dichas tareas deben ser enviadas a los nodos esclavos, cuando un nodo esclavo finaliza una tarea, inmediatamente realiza una solicitud al maestro para que asigne una nueva tarea. Esta tipo de técnica también es llamada como programación por demanda o bolsa de trabajo, y no sólo se aplica a problemas que tengan tareas que requieran de un mismo nivel de procesamiento.

Para los problemas con tareas que no presentan el mismo nivel de procesamiento, la mejor opción es dar prioridad y distribuir primero todas aquellas tareas que requieran una mayor carga computacional, lo ideal es dejar las tareas más complejas para el final, mientras que las tareas más pequeñas son atendidas por los nodos esclavos, al terminar pasan a modo espera sin hacer nada hasta que alguno otro esclavo diera por concluida su tarea más compleja, igual se puede utilizar esta técnica para otro tipo de problemas, por ejemplo donde el número de tareas no es uniforme y puede variar durante su ejecución.

Existen algunas aplicaciones, en las que una sola tarea al ser ejecutada puede generar un indeterminado número de tareas extras, pero al final el número de tareas pendientes debe reducirse a cero para así dar por finalizado el programa. Este tipo de situaciones suelen encontrarse en especial sobre los algoritmos de búsqueda.

Se puede hacer uso de una cola de peticiones para almacenar las tareas pendientes, pero si se diera el caso, todas las tareas mantienen un mismo tamaño y de la misma prioridad, una

cola FIFO suele ser más que suficiente, en otro tipo de situación es recomendable analizar y hacer uso la estructura que más se adecue a la situación.

3.2.2.2. Balanceo distribuido

Dentro del balanceo de carga distribuido intervienen un número N de nodos maestros los cuales se hacen cargo de un grupo con un número M de esclavos.

El balanceo de carga distribuido centralizado al igual que el balanceo de carga distribuido cuenta con una gran desventaja ya que el nodo maestro sólo tiene la capacidad de distribuir una sola tarea a la vez y después de haber terminado de distribuir todas las tareas iniciales sólo será capaz de responder a nuevas peticiones pero solo de una por una. Por lo que es muy probable que lleguen a producirse colisiones si dos o más esclavos hacen una solicitud de peticiones de manera simultánea.

Este tipo de estructura centralizada solo es recomendable si no se tiene la necesidad de hacer uso de muchos esclavos y las tareas requieren de gran capacidad computacional en el requerimiento de procesos, para tareas con un menor grado de requerimientos (tareas con muy poca carga computacional) y una mayor cantidad de esclavos es más apropiado distribuir las tareas en más de un sitio.

3.3. Balanceo de carga de trabajo en sistemas distribuidos

Es importante, ya que muchas de las aplicaciones distribuidas de manera paralela, por ejemplo, las que son utilizadas para la búsqueda o la optimización, les es sumamente difícil poder predecir el tamaño de las tareas que serán asignadas a cada dispositivo o computadora.

Es necesario que se realice una división del total de las tareas para que todas mantengan una carga uniforme y balanceada, cuando no se llega a presentar esta uniformidad, se dice que existe un desbalanceo en la carga y esto provoca que algunos procesadores se mantengan inactivos mientras el resto de ellos aún están calculando o realizando las tareas pendientes que se encuentran encoladas.

La mayoría de los procesadores tienen la necesidad de ser sincronizados al momento de la ejecución de algún programa, ya que si algún procesador no se encuentra en la misma situación que el resto entonces deberán esperar a que termine. Tener el conocimiento necesario sobre el balanceo de carga es importante ya que ayuda a lograr una distribución de forma equilibrada, la carga de trabajo entre todos los procesadores participantes y lograr la máxima velocidad de ejecución posible.

Para dar solución a una tarea o petición es necesario dividir en un número fijo de procesos que puedan ejecutarse de manera paralela, de esta forma cada proceso tendrá que realizar una cantidad estable de trabajos. Además, los procesos que se encuentran distribuidos entre todas las máquinas participantes, sin tomar en cuenta su tipo de procesador y velocidad, esto puede terminar en una situación muy particular en que algunos procesadores finalicen sus tareas mucho antes que el resto y a su vez queden libres, esto quiere decir que el trabajo no fue distribuido de forma equilibrada, porque algunos procesadores son más rápidos que otros, lo ideal sería que todos los procesadores trabajen de forma continua sobre las tareas disponibles para conseguir el mínimo tiempo de ejecución.

3.4. Balanceo de carga de trabajo en sistemas web

El balanceo de carga es un método utilizado principalmente para la distribución de tareas entre varias máquinas que se encuentran conectadas a una red. Un ejemplo es la distribución de las solicitudes HTTP entrantes (tareas) para una aplicación web en varios servidores. Hay una variedad de diferentes maneras de lograr llevar a cabo la implementación del equilibrio de carga, a continuación se mostraran algunos esquemas de equilibrio de carga común, junto a un diagrama que ilustra el principio básico de equilibrio de carga. Esto es visto en la figura 3.1.

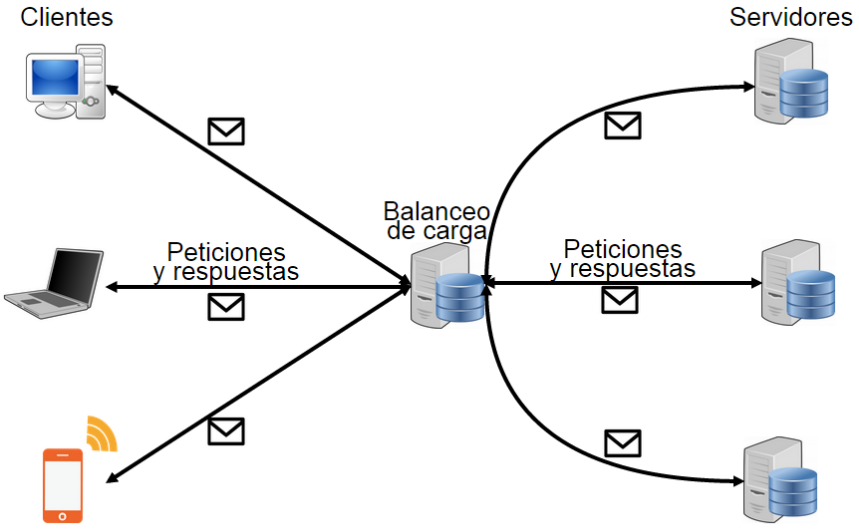

Figura 3.1: Balanceo de carga en sistemas web

El principal propósito del balanceo de carga [24] es distribuir la carga de trabajo de una aplicación en varios servidores, por lo que la aplicación puede procesar una carga de

trabajo superior, el balanceo de carga es una manera de escalar una aplicación. Un objetivo secundario que tiene el equilibrio de carga muy común (pero no siempre), es para proporcionar redundancia en la aplicación. Es decir, si un servidor dentro de un sistema de servidores falla, el equilibrador de carga puede eliminar temporalmente el servidor de la agrupación, y dividir la carga en los servidores que aún se encuentran en funcionamiento.

Tener múltiples servidores es de gran ayuda ya que se apoyan unos a otros, esto es lo que normalmente se denomina redundancia. Cuando ocurre un error, y las tareas se mueven desde el servidor fallido a un servidor que funcione, esto se suele llamar conmutación por error.

Un conjunto de servidores (clúster) que ejecutan la misma aplicación en cooperación se suele denominar como un "grupo"de servidores. El propósito de un clúster es típicamente cumplir los objetivos ya antes mencionados: para distribuir la carga en diferentes servidores, y para proporcionar redundancia/conmutación por error para el otro.

3.4.1. Distribución de carga de trabajo

Un esquema de la distribución uniforme de trabajo significa que las tareas se distribuyen de manera uniforme entre los servidores participantes de un clúster. Este esquema es, muy simple, hace que sea más fácil de implementar el esquema de la distribución de tareas, incluso también se conoce como Round Robin, es decir; los servidores reciben el trabajo en turnos rotativos (distribuida uniformemente). Aquí se muestra un diagrama que ilustra el equilibrio de carga, incluso la distribución de tareas (ver figura 3.2).

La distribución de carga de trabajo se dice que es adecuada cuando los servidores del clúster tienen la misma capacidad y las tareas enviadas estadísticamente requieren la misma cantidad de procesamiento.

La distribución de trabajo no toma en cuenta la diferencia que existe en el procesamiento que se requiere para procesar cada tarea, y aunque cada servidor se le dé el mismo rango de tareas, puede darse el caso que un servidor llegue a recibir un mayor número de tareas que requieran de una mayor cantidad de procesamiento que el resto. Esto pasa debido a la aleatoriedad de las tareas entrantes, ya que el servidor sobrecargado puede de un momento a otro recibir un conjunto de tareas o carga de trabajo más ligero o más pesado.

Así que, aunque el equilibrio de distribución de carga de trabajo continúa distribuyendo las tareas de manera uniforme en los servidores participantes, no se tiene la certeza de que se pueda dar lugar a una verdadera distribución uniforme de la carga de trabajo.

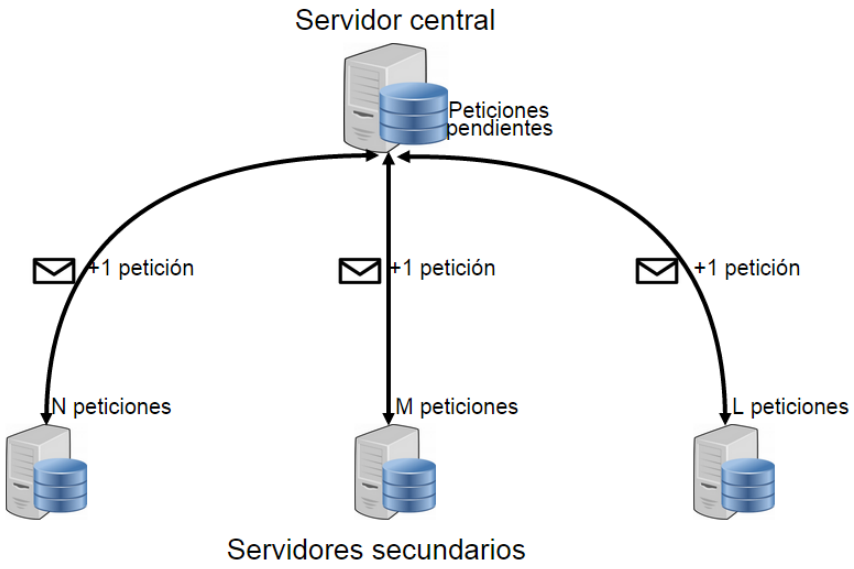

Figura 3.2: Distribución de carga de trabajo

3.4.2. Equilibrio de carga de trabajo por DNS

El equilibrio de carga por DNS está basado en un esquema simple, al configurarlo para devolver diferentes direcciones IP a diferentes equipos cuando solicitan una dirección IP para el nombre de dominio. Así se consigue un efecto que es similar al esquema de distribución de tareas, incluso con la excepción de que la mayoría de las computadoras almacenadas en caché con su dirección IP siguen llegando de nuevo a la misma dirección IP hasta que se realice una nueva búsqueda de DNS.

El equilibrio de carga basado en DNS es posible, aunque no es la mejor manera de mantener distribuido el tráfico de manera fiable a través de múltiples ordenadores, es mejor de usar la carga de equilibrio dedicada al software o hardware.

3.4.3. Distribución ponderada de tareas

Un esquema de equilibrio de carga de distribución ponderada de tareas, logra distribuir las tareas entrantes a los servidores del clúster basándose en el tamaño de la tarea, significa que se puede especificar el tamaño de las tareas de un servidor en relación con otros servidores, es muy útil sólo si la mayoría de los servidores del clúster tienen la misma capacidad.

Un ejemplo sería, si los servidores secundarios tiene diferentes capacidades de procesamiento, se pueden dividir las tareas en X, Y y Z, y así el primer servidor debería recibir X

tareas de acuerdo a la capacidad que tiene de procesamiento, el segundo servidor Y tareas, y el tercer servidor recibirá Z tareas, por cada N tareas recibidas de esta manera el servidor que cuenta con la menor capacidad de procesamiento recibe una menor cantidad de tareas en comparación con los otros dos y el que tiene la mayor capacidad de procesamiento recibe una mayor cantidad de tareas. En la figura 3.3 está un diagrama que ilustra este ejemplo.

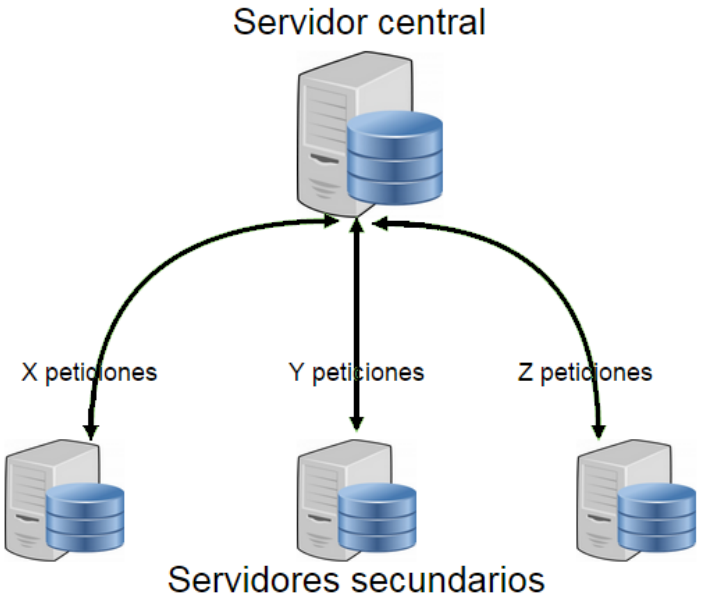

Figura 3.3: Distribución de tareas ponderada

El equilibrio de carga en la distribución ponderada de tareas es bastante útil cuando la mayoría de los servidores del clúster mantienen la misma capacidad, sin embargo, la distribución ponderada de tareas todavía no se basa en el trabajo necesario para procesar todas las tareas en consideración.

3.4.4. Sesión sticky

Los dos esquemas de equilibrio de carga mencionados anteriormente son basados en suponer que cualquier tarea entrante puede ser procesada de forma independiente de las tareas ejecutadas previamente, es verdad, aunque no siempre puede darse el caso.

Hay que darse a la idea de que si los servidores del clúster mantienen algún tipo de estado de sesión, como el objeto de la sesión en una aplicación web en java (o en PHP o ASP) y una tarea (solicitud HTTP) enviada al servidor 1 y este se traduce en escribir algún valor para el estado de sesión, ¿qué sucede si las solicitudes posteriores del mismo usuario se envían al servidor de 2 o 3 del servidor?, entonces ese valor de dicha sesión podría faltar, ya que este se almacena en la memoria del servidor 1. A continuación, se da un diagrama que ilustra dicha situación en la figura 3.4.

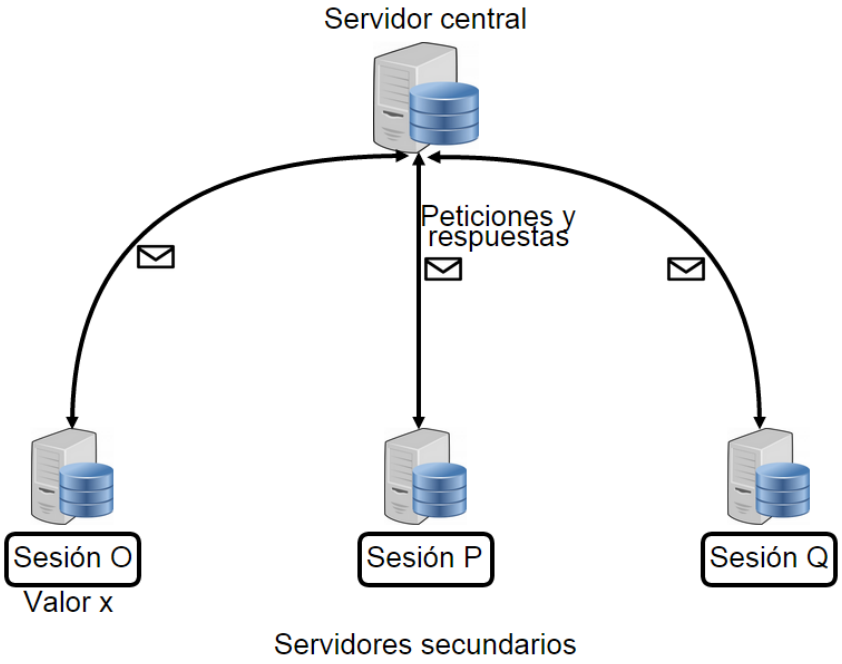

Figura 3.4: Sesión sticky

La mejor solución a este problema se conoce como sesión sticky de equilibrio de carga, todas las tareas (las solicitudes de HTTP) que son enviadas de la misma sesión (del mismo usuario) son recibidos por el mismo servidor, así de esta forma los valores de sesión almacenados podrían llegar a ser necesarios por las tareas posteriores (solicitudes) que están disponibles.

Con el equilibrio de carga de sesión no son las tareas las que se son distribuidas a los servidores, pero en lugar esto, nos da lugar a una distribución algo más imprevisible de la carga de trabajo, ya que algunas de las sesiones contendrán pocas tareas, y otras sesiones contendrá muchas más tareas.

Una solución más, es evitar completamente el uso de variables de sesión, o almacenar las variables de sesión en una base de datos o servidor de caché y las variables que tienen acceso a todos los servidores del clúster, es preferible evitar las variables de sesión completamente si es posible, excepto si existen razones para utilizar estas variables de sesión.

Otra forma de poder evitar las variables de sesión es con el uso de una interfaz gráfica de usuario con Aplicaciones de Internet Enriquecidas (RIA, Rich Internet Applications) más unas arquitectura que tenga la capacidad de soportar cualquier periodo de sesiones para variables de ámbito dentro de la memoria del cliente RIA, en lugar de que se encuentre dentro de la memoria del servidor web.

3.4.5. Distribución de tareas en cola

La distribución de tareas en cola, es muy similar en tamaño al esquema de distribución de tareas ponderado, pero con un toque especial, ya que en lugar de distribuir las tareas sin conocer su tamaño en los servidores del clúster, el equilibrio de la carga mantiene una cola de tareas para cada una de los servidores. Las colas de tareas contienen todas las peticiones que cada servidor que está procesando alguna tarea o que están en espera de alguna. A continuación se muestra diagrama que ilustra este principio (ver figura 3.5).

Cuando un servidor ha terminado una tarea y esta ha enviado de vuelta una respuesta HTTP a un cliente, la tarea es eliminada de la cola de peticiones para ese servidor.

Las tareas que aún se encuentran encoladas bajo los trabajos de esquema de distribución, aseguran de que cada cola de cada servidor tiene la misma cantidad de tareas correspondientes al mismo intervalo de tiempo, los servidores que cuentan con una mayor capacidad terminarán las tareas en un menor intervalo de tiempo a diferencia de los servidores con baja capacidad, y así, las colas de tareas de los servidores de mayor capacidad se liberaran más rápido y por lo tanto liberar un espacio para las nuevas tareas en un menor intervalo de tiempo.

Este esquema de equilibrio de carga lleva a cabo tanto el trabajo necesario para procesar cada una de las tareas y la capacidad de cada servidor, las nuevas tareas son enviadas a los

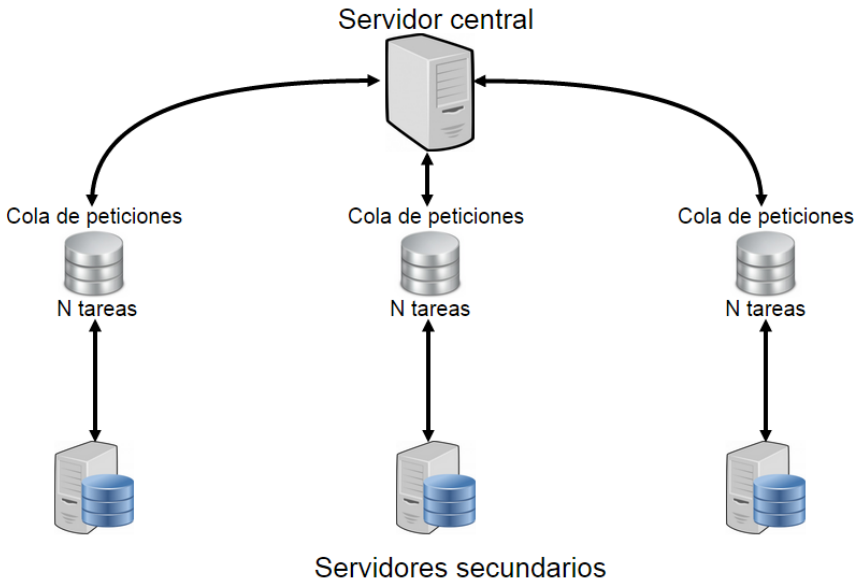

Figura 3.5: Distribución de tareas encoladas

servidores con el menor número de tareas encoladas, las tareas son eliminadas de las colas cuando estas han sido concluidas, lo que el tiempo que se tardó en procesar la tarea tiene un impacto de forma automática sobre el tamaño de la cola, dado que la velocidad con que se finaliza una tarea depende completamente de la capacidad del servidor ya que se toma automáticamente en cuenta.

Si un servidor se encuentra en sobrecarga temporalmente, su tamaño de la cola de tareas se convertirá en la más grande de las colas de tareas que el resto de los servidores del clúster, el servidor sobrecargado por lo tanto no tendrá la capacidad de recibir nuevas tareas que le sean asignadas a él hasta que ha reducido el número de tareas encoladas.

El encargado de equilibrar la carga tiene que realizar un mayor esfuerzo de lo que se representa en el uso de este esquema, hay que realizar un seguimiento de las colas de tareas y al mismo tiempo tiene que llevar un registro de cada momento en que es finalizada una tarea, por lo que debe ser eliminada de la cola de tareas correspondiente.

3.4.5.1. Cola autónoma

En este esquema de equilibrio de carga por medio de cola autónoma, todas las tareas entrantes son almacenadas en una cola de tareas, los servidores del clúster se conectan a esta cola y llevan el número de tareas que logran procesar. En la figura 3.6 se ilustra este esquema.

En este esquema no existe quien se encargue de equilibrar la carga real, cada servidor

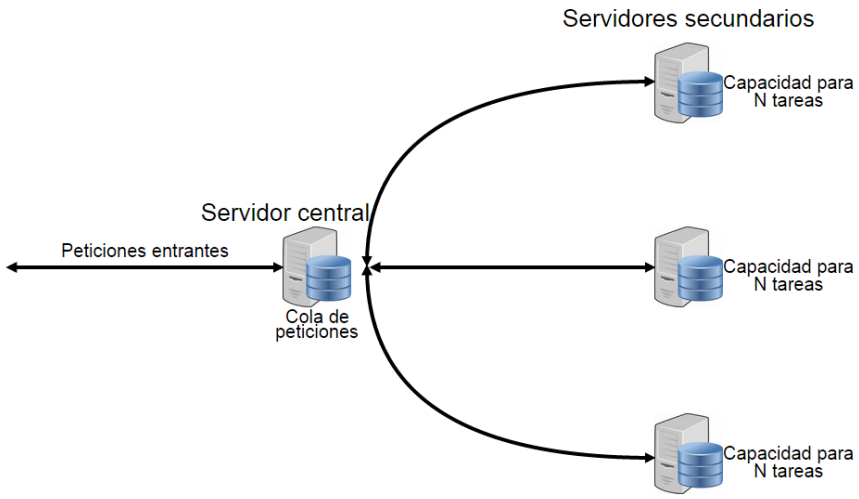

Figura 3.6: Balanceo por cola

tiene específicamente la cantidad de tareas que es capaz de realizar, no es sólo la cola de tareas y el servidor. Si un servidor de la agrupación llegara a fallar, sus tareas se mantienen en estado natural dentro de la cola de tareas, y estas son procesadas por otros servidores después, por esta razón es muy necesario que cada una de las funciones de los servidores se mantenga de forma autónoma dentro de los otros servidores y de la cola de tareas.

El responsable de balancear la carga necesita saber qué servidores son parte del clúster, la cola de tareas no tiene la necesidad de saber acerca de los servidores, cada servidor sólo necesita saber acerca de la cola de tareas. El equilibrio de carga de colas autónoma mantiene implícita la carga de trabajo y la capacidad de cada corte de tareas, los servidores sólo toman las tareas de la cola y tienen la capacidad de realizarlas.

La cola autónoma tiene desventajas en comparación con la distribución del tamaño de la cola, un servidor que requiera una tarea necesita mantener una comunicación primero con la cola, después descargar la tarea y al final regresar una respuesta, este proceso realiza alrededor de 2 a 3 viajes redondos dentro de la red (en función de si una respuesta tiene que ser enviada de vuelta).

El esquema de distribución de tamaño y de colas tiene un recorrido de red menor, ya que el responsable de equilibrar la carga envía una solicitud a un servidor y este devuelve una respuesta (si es necesario). Eso es sólo una ida y una vuelta sobre la red.

3.4.6. Balanceo por software

Balanceo de carga por software se puede proporcionar de dos maneras: 1: Como parte de un sistema operativo o como una aplicación complemento, el balanceo de carga es implementado como una solución basada en software que comúnmente ofrece una facilidad de despliegue y mantenimiento, al igual que mantiene un rendimiento similar a las soluciones basadas en hardware. Algunas implementaciones muy comunes de balanceo de carga basado en el software incluyen aquellos que se encuentran dentro de Microsoft Windows o Linux.

- Perlbal: es un servicio gratuito basado sobre el balanceo de carga de aplicaciones, ofrece apoyos muy parecidos al balanceo HTTP a varios servidores back-end sobre la marcha, esta es una de las principales características que logran definir a Perlbal ya que puede configurar o reconfigurar sobre la marcha sin la necesidad de realizar un reinicio de software.
- Un equilibrador de carga es un servidor (o dispositivo) que tiene la tarea de equilibrar las solicitudes recibidas a través de un número N de servidores participantes para distribuir la carga total, un proxy inverso puede hacer esto, pero también tiene otras características.
- Pound: es un proxy inverso, utilizado como balanceador de carga y HTTPS front-end para servidores web, este fue desarrollado principalmente para permitir la distribución de la carga entre varios servidores de web y tener un mejor soporte SSL para los servidores web que no lo ofrecen de forma nativa. Pound es distribuido bajo la licencia GPL la cual no ofrece ninguna garantía ya que es completamente gratuita.
- Linux Virtual Server (LVS): es un sistema de alta disponibilidad y altamente escalable para el manejo de clústers de servidores reales. Por diseño, se mantiene una arquitectura transparente para el usuario final, lo que permite al usuario interactuar con él como si se tratara de un único servidor. La escalabilidad del sistema se logra gracias a la transparencia al momento de agregar o quitar nodos al clúster, la alta disponibilidad es proporcionada por la detección de nodos o bien por fallas de algún dominio, y la reconfiguración del sistema de forma adecuada.

3.4.7. Balanceo por hardware

La forma más fácil de lograr un balance de solicitudes entre varias máquinas interconectadas entre sí, es utilizar un dispositivo hardware, sin embargo hay que realizar un ajuste necesario a la configuración y empezar a distribuir el tráfico, existen desventajas al ser utilizado un dispositivo hardware. Una de ellas es la configuración ya que puede llegar a ser bastante complicada.

Balanceo de carga por medio de un enrutador: hay varias rutas al mismo destino con la misma capacidad, es razonable utilizar a todas las rutas y distribuir el tráfico en partes iguales entre todos los nodos, los nodos alternativos pueden garantizar un uso libre de bucles, ya que son simétricos con respecto a la capacidad de la ruta principal. Si hay varias rutas con capacidades diferentes, esta idea no podía ser de fácil aplicación, como se muestra en la figura 3.7:

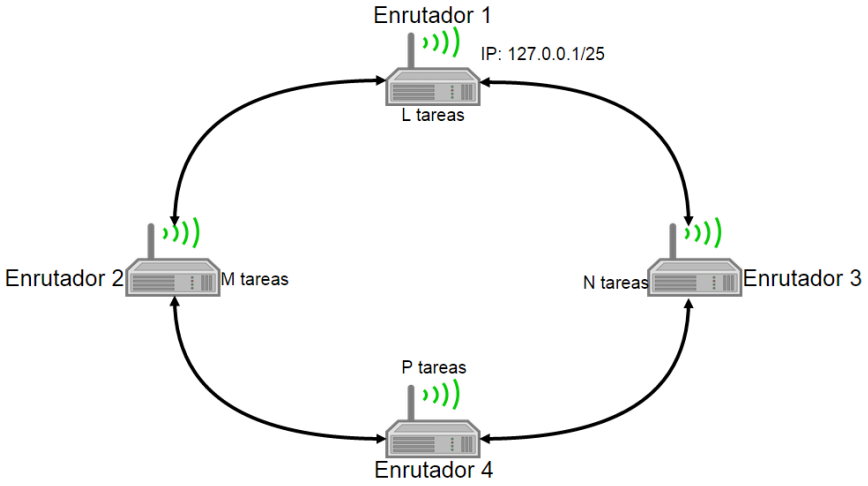

Figura 3.7: Balanceo por hardware

Si el enrutador recibe e instala varias rutas con la misma distancia y capacidad a un destino, se dice que puede ocurrir balanceo de carga, el número de rutas de acceso utilizadas están limitadas por el número de entradas que el protocolo de enrutamiento pone en la tabla de enrutamiento, cuatro entradas es el valor por defecto en IOS para la mayoría de protocolos de enrutamiento IP, con la excepción del protocolo de entada externa (BGP, por sus siglas en Inglés Border Gateway Protocol), donde una entrada es el valor predeterminado.

3.5. ¿Qué es la distribución de carga de trabajo?

Este en uno de los principales métodos para que los sistemas distribuidos logren obtener el mayor grado de eficiencia posible, al dispersar el trabajo a realizar sobre todas las máquinas participantes dentro de la red, esto sin mantener un equilibrio de tareas asignadas a cada máquina.

Esta consiste en la distribución de la carga [25] entre los diferentes nodos del sistema para obtener la mayor eficiencia posible.

3.6. Distribución de carga de trabajo en sistemas distribuidos

La carga se distribuye y trasladada de los nodos con mayor carga de trabajo a los nodos con menor carga dentro del sistema, con esto se logra obtener una mayor reducción de tiempo en la ejecución de las tareas dentro del sistema distribuido y se obtiene una proximidad del tiempo de conclusión sobre la ejecución de las tareas entre cada uno de los nodos, en otras palabras, se consigue que todos los nodos terminen de ejecutar sus tareas en un mismo intervalo de tiempo, y que no existan un rango mayor de tiempo entre la finalización las tareas entre cada uno de ellos y el resto de los nodos participantes.

Así se logra conseguir que no existan nodos con sobrecarga, y otros totalmente libres de trabajo alguno, ya que estos nodos sobrecargados en automático compartirían parte de su trabajo a los nodos que se encuentran libres de trabajo.

Los algoritmos de distribución de carga se basan en unos componentes que marcan las diferencias entre unos algoritmos y otros:

- Política de información: Decide ¿Qué tipo de información debe ser adquirida de los nodos?, ¿De qué nodos debe recoger ésta información?, y ¿Cuándo debe recogerla?
- Política de selección: Decide cuál es la carga que se va a ser transferidas dentro de un proceso de transferencia.
- Política de localización: Localiza el nodo más apto para la realización de una determinada transferencia.
- Política de transferencia: Decide que nodos son los más aptos o no para llevar a cabo una transferencia, tanto de emisor como de receptor.

3.6.1. Distribución de procesos

Es una técnica en la cual se distribuye la información a través de un número N de máquinas participantes, estas máquinas tienen la posibilidad de ser tanto ordenadores como terminales de datos con un cierto nivel de inteligencia, las máquinas están conectados sobre una red de comunicación.

Un sistema distribuido básicamente es cuando los componentes se encuentran situados en ordenadores que están conectados mediante una red comunicándose y manteniendo una coordinación de sus acciones sólo por la transmisión de mensajes.

Por ejemplo:

- El internet, cuando los servidores se encuentran globalmente situados en diferentes partes.
- Una intranet, que es una porción de la Internet gestionada por una organización y que sus servidores se encuentran situados dentro de un área local.

3.6.1.1. Los requisitos de diseño para las arquitecturas distribuidas

- Balancear las cargas informáticas, para evitar un posible colapso.
- Mantener un buena calidad en los servicios, para que los usuarios se sientan seguros con la fiabilidad y prestaciones que ofrece el sistema.
- El almacenamiento en caché y replicación, para tener un acceso mucho más rapido a la información que es solicitada con más frecuencia.
- Los problemas de rendimiento frecuentemente parecen ser muy importantes para una implementación exitosa de sistemas distribuidos, pero han sido mucho las avances en el diseño de estos sistemas que los superados por el uso constante de la replicación de datos y el almacenamiento en caché.
- Seguridad, se hace uso de protección pasiva y activa, junto con mecanismos de serguridad criptográfica para evitar probables ataques o uso indebido de recursos.
- La tolerancia a fallas, se consigue con el uso de redundancia en software y hardware, ayuda a que el sistema continúe con su funcionamiento correcta en presencia de fallas en software, hardware o sobre la red.

3.6.1.2. Formas de procesamiento distribuido

- Aplicaciones distribuidas: una aplicación se puede dividir en varios componentes los cuales se dispersan entre un número n de máquinas. Una aplicación es replicada sobre un número n de máquinas conectadas entre sí. Un gran variedad de distintas aplicaciones son distribuidas entre un número n de máquinas participantes. Puede ser caracterizado por una partición de manera vertical u horizontal.
- Dispositivos distribuidos: son un conjunto de dispositivos de manera distribuida que es posible controlar mediante los procesadores, como son, los cajeros automáticos o equipos de interfaz de algún laboratorio. Es la distribución de procesamiento tecnológico enviada a varios lugares para el proceso de fabricación en la automatización de fábricas.

3.6.1.3. Técnicas de procesamiento distribuido

1. Centralizado: como su nombre lo dice se realiza en una ubicación central, utiliza a los terminales que se encuentran conectados a una única computadora central, la cual realiza las funciones de cómputo y controla los distintos terminales de manera remota. Este sistema se basa en su totalidad sobre la computadora central como se observa en la figura 3.8.

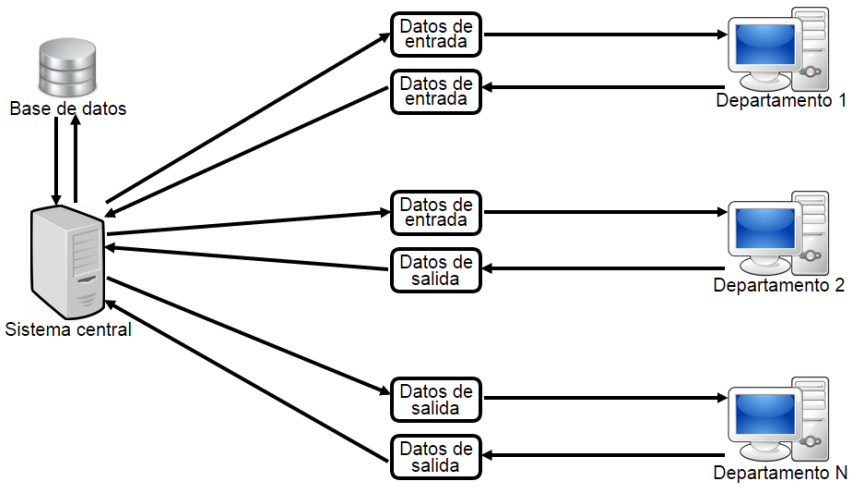

Figura 3.8: Procesamiento centralizado

2. Descentralizado: es cuando los sistemas informáticos se localizan en diferentes lugares, a pesar de que los datos sean transmitidos entre varios ordenadores periódicamente (ver figura 3.9).

3. Paralelo: es un procesamiento simultáneo que realiza la misma tarea sobre uno o más microprocesadores con el fin de obtener resultados más rápidos. O bien, un procesador múltiple o equipo múltiple con recursos de memoria compartida como se aprecia en la figura 3.10.

4. Distribuido abierto: ODP, por sus siglas en Inglés (Open Distributed Process), es un modelo de referente a las computadoras que proporciona un margen de coordinación para lograr una normalización sobre el procesamiento distribuido abierto. Esta tiene compatibilidad con la distribución, el inter funcionamiento, la plataforma y la independencia tecnológica y la portabilidad en conjunto con un marco sobre la arquitectura de la empresa para la especificación de sistemas de ODP.

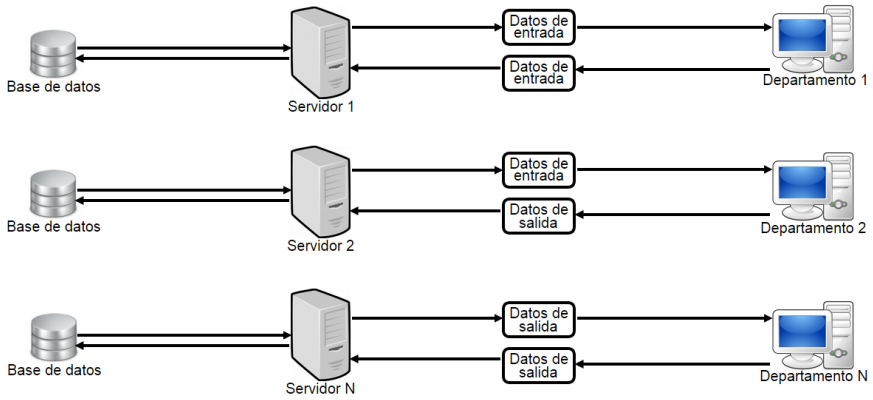

Figura 3.9: Procesamiento descentralizado

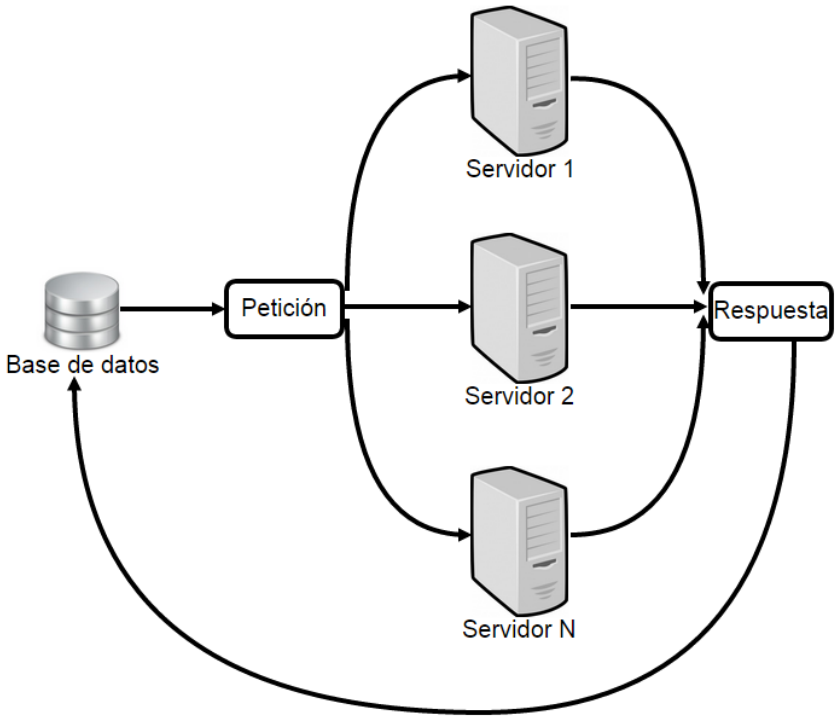

Figura 3.10: Procesamiento paralelo

Capítulo 4

Replicación de información

Al momento de distribuir información en un sistema distribuido, se busca garantizar un óptimo funcionamiento en las bases de datos, gracias a la disponibilidad que se obtiene propagando la información en varios nodos o máquinas participantes. Todos los aspectos a ser considerados para tal efecto serán descritos como sigue.

4.1. ¿Qué es la replicación de información?

Antes de replicar una base de datos, toda técnica que se decida emplear debe ser con base en las ventajas y desventajas que ofrece. Además, se debe saber elegir la herramienta sin importar el sistema gestor que se esté ocupando para administrar dicha base de datos (ver figura 4.1).

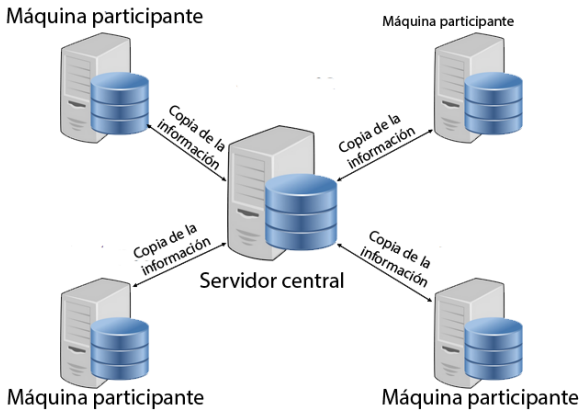

Figura 4.1: Replicación de información

La replicación de información [26] es un mecanismo utilizado para propagar y diseminar datos en un ambiente de sistemas distribuidos, con el objetivo de tener mejor rendimiento y confiabilidad en los sistemas en los que sean implementados, mediante la reducción de dependencia de un sistema de base de datos centralizado.

La replicación de datos es utilizada para el mantenimiento automático de copias de datos múltiples en las computadoras, es una técnica utilizada para la mejora de los servicios que nos ofrecen los sistemas distribuidos porque proporciona una mejora del rendimiento de los servicios e incrementa su disponibilidad haciéndolo tolerante a fallos.

El objetivo de la replicación es que las operaciones que realizan de los clientes sobre las réplicas se realicen de forma consistente y con un tiempo de respuesta satisfactorio, algunas veces la consistencia no se logra debido a que las réplicas de algún cierto objeto no son necesariamente idénticas y en algunas ocasiones las réplicas pudieron haber recibido actualizaciones que otras no las hayan recibido.

4.2. ¿Por qué replicar una base de datos?

Es conveniente replicar una base de datos cuando es usada en un ambiente de sistemas distribuidos, ya que implica un compartimiento de toda la información entre varias computadoras o servidores, las cuales pueden tener acceso aún sin estar en el mismo lugar de manera remota.

La replicación también es usada para poder aumentar el paralelismo entre las consultas de la base de datos, ya que toda la información estará siempre disponible en más de una localidad de la red, ya sea un servidor o una computadora local que siempre la contenga.

También ayuda a mejorar la disponibilidad de los datos ante posibles caídas del servidor sin sufrir perdida de datos, esto es muy importante ya que si un nodo o servidor sufre un desperfecto no se tendrá perdida de información porque toda la información se encuentra replicada en todos los demás nodos del ambiente de sistemas distribuidos y se podrá acceder a ella sin problema.

4.3. Técnicas de replicación

De entre las técnicas de [27] existentes, se busca un equilibrio entre la consistencia, el rendimiento y la complejidad.

Existen dos modelos de replicación: pasiva y activa; Cada una tiene una manera diferente de funcionamiento y las dos están divididas en fases y análisis que serán dettaladas a continuación.

4.3.1. Activa

En la figura 4.2 se muestra el funcionamiento de la replicación pasiva o primaria, la cual tiene a la vez uno o más respaldos a su disposición, por si acaso llegará a fallar uno de estos respaldos actué como primario, los clientes solo pueden comunicarse con el respaldo primario.

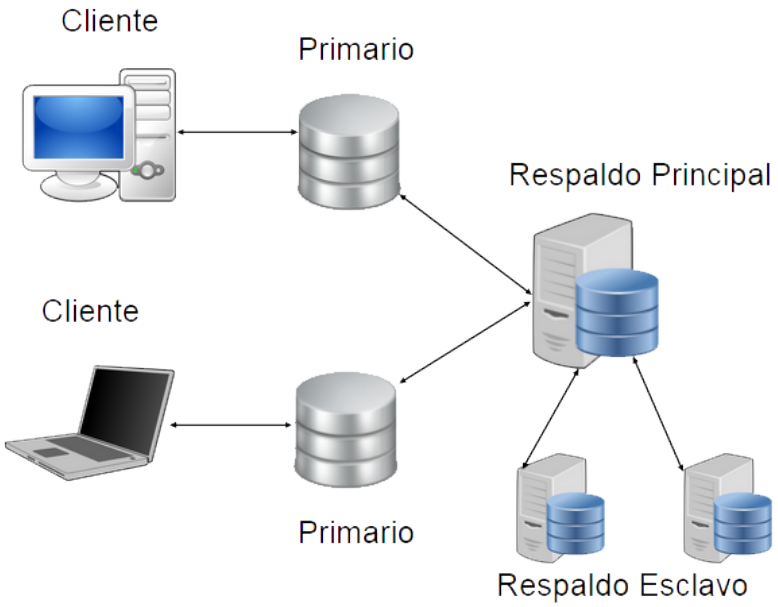

Figura 4.2: Replicación activa

Etapas de Replicación Activa

- El cliente envía la petición hacia el gestor primario.
- El gestor primario ejecuta las peticiones que se le ordenan siguiendo un método de ordenación denominado FIFO.

- Se ejecuta la petición y se guarda la respuesta
- El gestor primario envía la respuesta hacia el cliente.

La Replicación Activa tolera fallos de proceso, el cliente requiere de poca funcionalidad al controlar el orden de modificación mediante el gestor primario, mantiene la linealizabilidad.

Su único problema de esta replicación es que sufre de cuellos de botella, ya que antes de que se dé una respuesta es necesario actualizar los respaldos, atrasando a esta.

4.3.2. Pasiva

En la replicación pasiva no es como la activa, ya que aquí todos los gestores son iguales, los clientes reparten todas las peticiones hacia los gestores. La figura 4.3 muestra las etapas de la replicación pasiva.

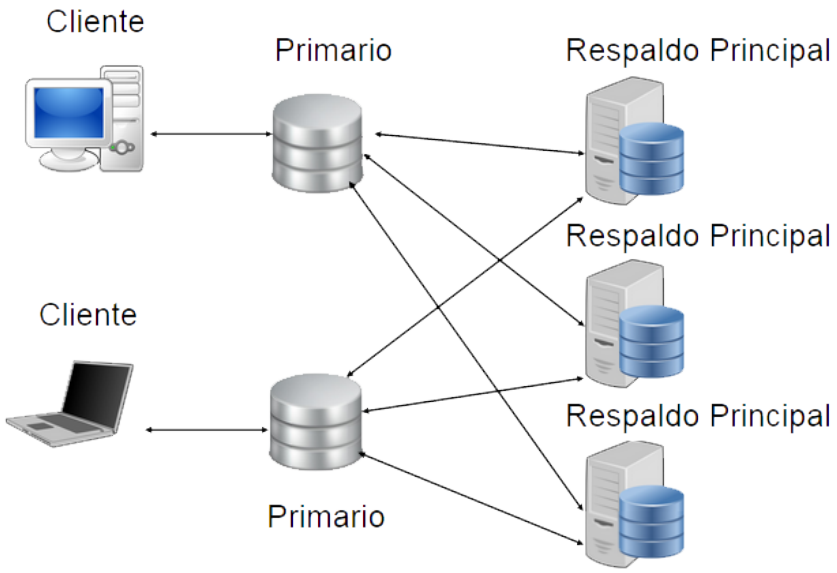

Figura 4.3: Replicación pasiva

- El cliente reparte las peticiones hacia los gestores y este no envía otra petición hasta que la anterior haya sido respondida.
- El sistema coordina la comunicación entre los demás gestores entregando la petición.

- Cada gestor ejecuta una petición.
- Cada gestor envía una respuesta al cliente, el cliente recoge la respuesta dependiendo de los fallos que se hayan producido durante la respuesta.

La replicación Pasiva se basa en la multidifusión fiable, que es la que se encarga totalmente de ordenar la aportación a la tolerancia a fallos.

4.4. Tipos de replicación

Existen dos tipos de replicación de información que son los más usados, la replicación completa y replicación parcial, cada tipo de replicación será explicado a detalle en los siguientes puntos.

4.4.1. Replicación completa

En este tipo de replicación la base de datos guarda varias copias de cada fragmento que serán guardadas en varios sitios, siendo así los fragmentos estarán replicados. Cuando una base de datos está totalmente replicada puede que no sea practica debido a que le impone una gran carga al sistema y puede reducir drásticamente la velocidad de las operaciones de la actualización, ya que cada que se ejecute una actualización lógica deberá de ejecutar en todas y cada una de las bases de datos con tal de mantener la consistencia en ellas (ver figura 4.4).

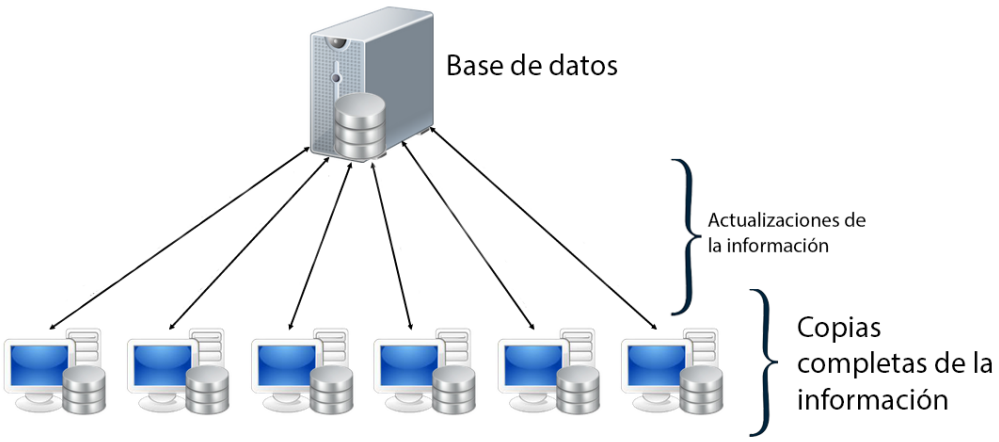

Figura 4.4: Replicación completa

4.4.2. Replicación parcial

Este tipo de replicación lo que hace es qué un administrador decide que conjunto de datos van a ser replicados y en que sitios estarán ubicados, a partir de aquí todos los nodos o servidores conocen el esquema de replicación que se estará usando y se comportan siguiendo algún algoritmo de control de concurrencia. Este tipo de replicación se aprecia en la figura 4.5.

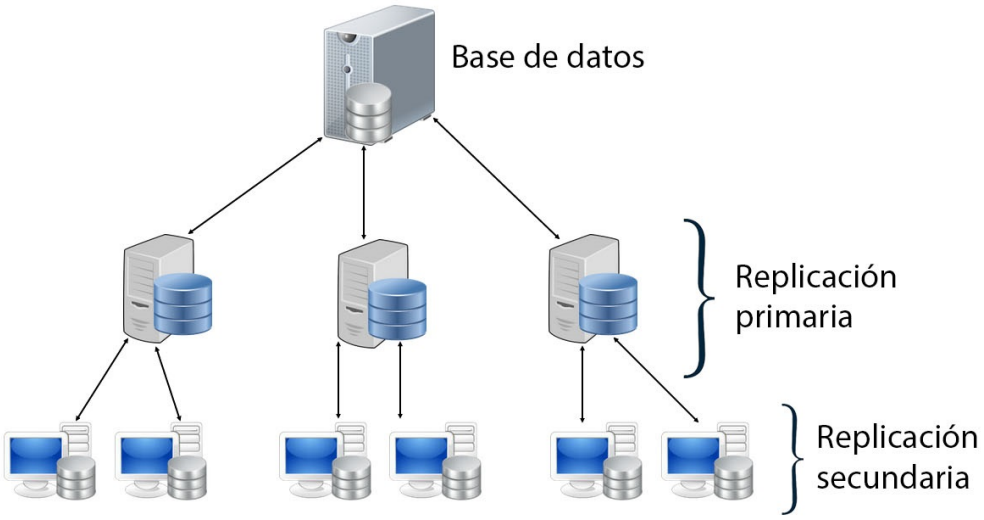

Figura 4.5: Replicación parcial

4.5. Aspectos a tomar en cuenta

Algunos aspectos que se deben de tomar en cuenta [28] a la hora de optar por la replicación de las bases de datos son:

- El tamaño de la base de datos, se tiene que tomar en cuenta si la base de datos que será replicada es pequeña o grande ya que aquí influirán los costos de desarrollo y si es o no recomendable optar por la replicación
- La frecuencia con la cual será usada, si su uso es elevado de forma remota y la base de datos es grande, la replicación reducirá el costo de las solicitudes que se harán hacia los datos.

- Costos de desarrollo, son costosos ya que se deben de que software será el que se ocupara, el rendimiento que debe de tener la base de datos y los costos de administración del sistema
- Autonomía, la autonomía de un servidor da la medida de cuanto puede operar el sitio desconectado de la base de datos principal.
- Latencia, en la replicación la latencia se refiere al momento en que se deben de sincronizar todas las copias de los datos en todos los nodos involucrados.

4.6. Ventajas y desventajas

- Disponibilidad: Cuando un servidor de la red llegue a fallar no se sufre de perdida de información ya que se encuentran alojados más backups en diferentes sitios o servidores, esto no pasaría si la base de datos solo estuviera alojada en un solo servidor.
- Fiabilidad: Al estar la información alojada en varios servidores se convierte en un gran sistema para recuperar la información cuando existan fallos en alguno de los servidores.
- Se Incrementa el paralelismo: Una ventaja favorable es el paralelismo con el cual trabaja, que mientras más sitios haya, mayor será la disponibilidad de que los datos que son necesarios se hallen en el sitio donde está siendo ejecutada la transacción; esto se resume en una minimización de los movimiento que se llevan a cabo entre los datos de los sitios.

- Sobrecarga incrementada durante la actualización: Una desventaja es que cada que una de las réplicas debe de ser actualizada para así evitar fallos, así que mientras más grande sea la red de sitios mayor tiempo tomara para ser actualizada. Un claro ejemplo son los bancos, en cada sucursal tienen que tener siempre la misma información actualizada del cliente.
- Costos en desarrollo de software: Es más caro desarrollar un sistema de bases de datos distribuidas.
- Mayor tiempo en procesamiento: El intercambio de mensajes y cálculos adicionales son una forma de tiempo mayor que no existe en los sistemas centralizados.

4.7. Herramientas existentes para los SGBD más comunes

A continucación se mencionarán las herramientas más ocupadas para la replicación de bases de datos:

4.7.1. Rubyrep

Esta herramienta es usada por MySQL y Postgres, la cual está escrita en JRuby haciéndola una herramienta independiente y multiplataforma, esto quiere decir que en donde se pueda ejecutar Java, Rubyrep lo hará sin problemas, puede hacer una sincronización de dos bases de datos y soporta replicación asíncrona, Master-Master, Master-Slave [29].

4.7.2. Slony-I

Es ocupada por PostgreSQL y es un sistema diseñado para uso en centros de datos y sitios de respaldo, donde el modo normal de operaciones es donde todos los nodos están disponibles, soporta PostgreSQL, nos permite realizar replicaciones maestro/esclavo asíncronos, realizando actualizaciones en cascada, además Slony-I está disponible para las plataformas Linux y Windows [30]

4.7.3. SymmetricsDS

Es soportada por un número mayor de gestores de bases de datos como lo son Oracle, MySQL, PostgreSQL. SymmetricsDS es un software de replicación de datos asíncrona que permite subscriptores múltiples y sincronización bidireccional. Utiliza tecnologías web y de base de datos para replicar tablas entre bases de datos relacionales, casi en tiempo real. SymmetricsDS está escrito en el lenguaje Java, lo cual lo hace multiplataforma y cuenta con licencia de código abierto GPL [31].

4.7.4. Pgpool-ll

Es un middleware que trabaja entre Servidores PostgreSQL y bases de datos PostgreSQL, está autorizado bajo una licencia BSD, esta herramienta ayuda en la replicación y el balanceo de carga, así como la paralelización de consultas solo en el gestor de base de datos PostgreSQL,

y es soportado únicamente por las plataformas Linux, Solaris, FreeBSD y UNIX quedando fuera Windows [32].

4.7.5. BUCARDO

Es una herramienta de replicación de datos para el SGBD PostgreSQL. Este sistema es asíncrono, se puede realizar sincronización de tipo maestro-esclavo y maestro-maestro, siendo desarrollado con el lenguaje de programación Perl y su secreto para ser asíncrono es PL/Perl. Es Software Libre y esta liberado con la licencia BSD. Solo funciona con PostgreSQL, es estable y fácil de usar [33].

4.7.6. Daffodil Replicator

Es otra alternativa más de herramientas para la replicación de bases de datos, ya que posee una lista grande de gestores de bases de datos, tales como: SQL Server, Oracle, MySQL, PostgreSQL. Haciendola una herramienta de código libre la cual está escrita en el lenguaje Java para la integración de datos, migración y protección en tiempo real de los mismos. Esto permite la replicación de datos bi-direccional y la sincronización entre bases de datos homogéneas y heterogéneas. Daffodil Replicator está disponible para Linux, Mac y Windows [34].

Capítulo 5

Fragmentación de información

Con la fragmentación de las bases de datos se tendrá la ventaja de alojar fragmentos de estas en diversos servidores, trabajando así de forma horizontal, vertical o de forma mixta ya que cada tipo de fragmentación trabaja de forma diferente, se detallará el desempeño y la facilidad de administración que tiene cada una y los requerimientos que se deben tener en cuenta a la hora de fragmentar la base de datos, así como la replicación la fragmentación también tiene ventajas y desventajas cuando se quiera implementar.

5.1. ¿Qué es la fragmentación de información?

La fragmentación es un conjunto de técnicas para dividir la base de datos en unidades lógicas, llamadas fragmentos, cuyo almacenamiento puede asignarse en diversos sitios. Estas técnicas se utilizan durante el proceso de diseño de bases de datos distribuidas. La información concerniente a la fragmentación de los datos se almacena en un catálogo global del sistema al que tiene acceso el software cliente cuando es necesario (ver figura 5.1).

El objetivo de la fragmentación consiste en dividir una relación en un conjunto de relaciones más pequeñas tal que algunas de las aplicaciones de usuario sólo hagan uso de uno de esos pequeños fragmentos, una fragmentación óptima es aquella que produce un esquema de división que minimiza el tiempo de ejecución de las aplicaciones que emplean esos fragmentos. [35]

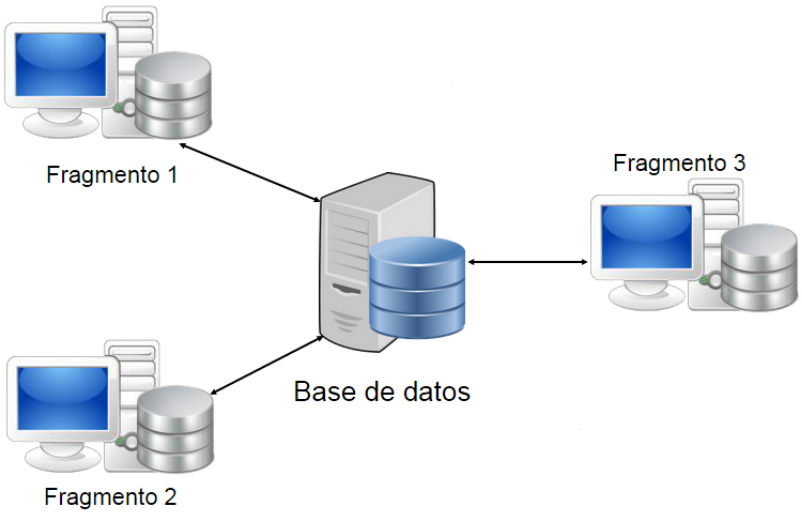

Figura 5.1: Fragmentación de información

5.2. ¿Por qué fragmentar una base de datos?

La fragmentación se diseña teniendo en cuenta el uso que se le va a dar, utilizando los datos que se almacenan en cada una de las sedes, construyendo relaciones más pequeñas y más adaptadas a las operaciones de recuperación y actualización que utilizan las aplicaciones, tratando así de aligerar las comunicaciones entre los nodos debido al alto costo que éstos tienen [36].

Las cuatro razones para la fragmentación son las siguientes:

- Es útil ya que las aplicaciones de base de datos suelen funcionar con vistas y por ello se pueden utilizar distintas relaciones en diferentes nodos para formar la unidad distribuida.
- Se consigue una mayor eficiencia dado que los datos suelen estar almacenados cerca del nodo que más utiliza dichos datos.
- Permite aumentar el grado de concurrencia porque la fragmentación de las relaciones permite que una transacción pueda dividirse en subconsultas que operan sobre estos fragmentos.

- Aporta una mayor seguridad, dado que los datos no utilizados por un nodo local no se almacenan en él y por lo tanto no están al alcance para personas sin autorización.

5.3. Grado y reglas de fragmentación

El grado de fragmentación antes de que sea aplicada en una base de datos es una decisión importante ya que afecta el rendimiento en la ejecución de las consultas, el grado en que se dará la fragmentación tiene dos extremos; no fragmentar nada o fragmentar de tal manera que sea un nivel de tuplas individuales esto ya sea en caso de una fragmentación horizontal o a nivel de atributos individuales en caso de una fragmentación vertical.

También tiene que tomarse en cuenta el grado en que va a tener la fragmentación ya que dependerá de las necesidades de la aplicación en donde estará alojada y corriendo la base de datos.

Ya que se tiene en cuenta que la fragmentación podría afectar en el rendimiento de nuestra base de datos en especial cuando es usada en distintos fragmentos que están ubicados en diferentes nodos, para poder asegurar que la base de datos no tendrá cambios semánticos cuando se esté llevando a cabo la fragmentación; se deben de tomar en cuenta estas 3 reglas:

- Completitud: esta referida a una descomposición sin pérdida de datos, es decir que se asegure que los datos en una relación global pueden ser mapeados en fragmentos sin ningún tipo de perdida.
- Reconstrucción: la posibilidad de reconstrucción de una relación global a partir de fragmentos de relaciones asegura que las restricciones definidas sobre los datos en forma de dependencias sean preservadas.
- Disyunción: si la relación se particiona de forma horizontal de un dato que se tenga en un fragmento, no debe de estar presente en algún otro fragmento de la relación.

5.4. Tipos de fragmentación

La fragmentación horizontal, vertical e híbrida [37] son aspectos importantes del diseño físico de base de datos que tiene un impacto considerable en el desempeño y la facilidad de administración de bases de datos relacionales y bases de datos orientadas a objetos.

Al igual que los índices y las vistas materializadas, los tres tipos de fragmentación pueden impactar el desempeño de una carga de trabajo (conjunto de consultas) reduciendo el costo de acceder y procesar datos.

5.4.1. Fragmentación horizontal

Como se ha explicado anteriormente, la fragmentación horizontal se realiza sobre las tuplas de la relación como lo muestra la figura 5.2. Cada fragmento será un subconjunto de las tuplas de la relación. Existen dos variantes de la fragmentación horizontal: la primaria y la derivada. La fragmentación horizontal primaria de una relación se desarrolla empleando los predicados definidos en esa relación. Por el contrario, la fragmentación horizontal derivada consiste en dividir una relación partiendo de los predicados definidos sobre alguna otra.

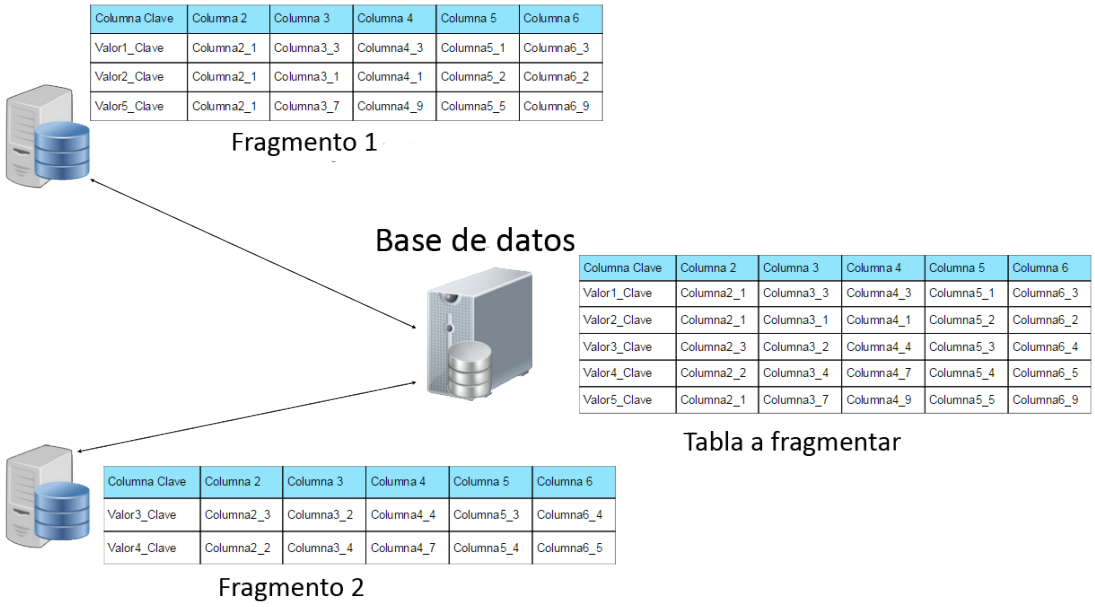

Figura 5.2: Fragmentación horizontal

La fragmentación horizontal divide una relación "horizontalmente" agrupando filas para crear un subconjunto de tuplas, donde cada subconjunto tiene un cierto significado lógico, estos fragmentos pueden entonces asignarse a diferentes nodos en el sistema que se está distribuyendo.

Requerimientos.
Información acerca de la base de datos: concierne a la información del esquema conceptual global de la base de datos, en este contexto es importante notar cómo están conectadas las relaciones de la base de datos una con otra.

Las conexiones entre los objetos de la base de datos, en este caso llamadas relaciones, deben ser absolutamente familiares para los que han manejado modelos de sistemas de bases de datos.

La relación al final de la conexión es llamada propietario de la conexión y la relación que está en la cabeza de la conexión es llamada miembro.

5.4.2. Fragmentación vertical

La fragmentación vertical divide una relación "verticalmente" en columnas como lo demuestra la figura 5.3, así cada fragmento mantiene algunos atributos específicos de la relación original. Este tipo de fragmentación produce fragmentos que contienen a un subconjunto de atributos así como las claves principales de la relación, el objetivo principal de la fragmentación vertical es particionar una relación en un conjunto de relaciones más pequeñas de tal manera que las aplicaciones de los usuarios corran en un solo fragmento.

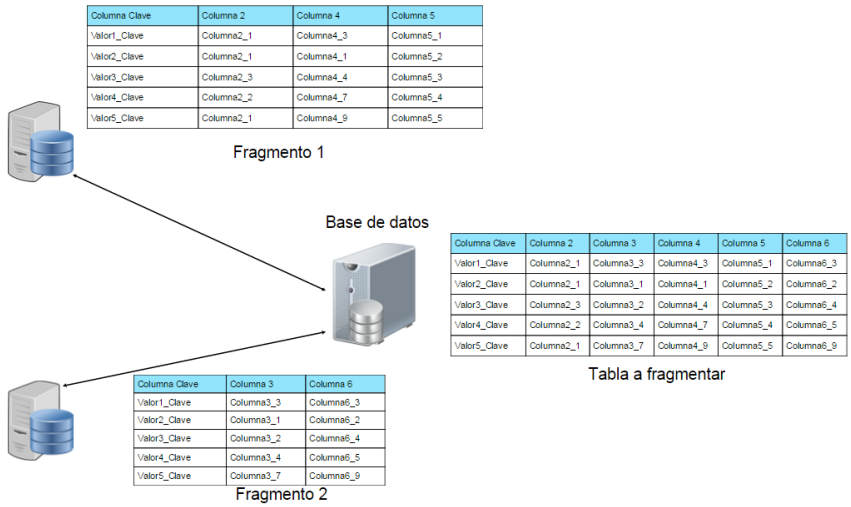

Figura 5.3: Fragmentación vertical

Cabe mencionar que este tipo de fragmentación no es del todo adecuada ya que si ambos fragmentos son almacenados por separado no se podrán volver a juntar las tuplas originales, ya que no hay un atributo común entre ambos fragmentos.

Requerimientos.
La información principal que es requerida para la fragmentación vertical es la que esta relacionada con las aplicaciones, ya que en el particionamiento vertical se coloca en un fragmento los atributos a los cuales serán accesados juntos.

5.4.3. Fragmentación hibrida y mixta

La fragmentación hibrida o mixta es una unión de las fragmentaciones horizontal y vertical en una sola relación, el funcionamiento de este tipo de fragmentación se divide en una relación con un conjunto de fragmentos los cuales cada uno contiene un subconjunto de atributos y de tuplas de la relación, esto se logra ejecutando la fragmentación horizontal y después la vertical o viceversa, que producirá un árbol de particionamiento estructurado, ya que estos dos tipos de particionamiento que se tiene serán aplicados uno después del otro. Este tipo de fragmentación se aprecia en la figura 5.4.

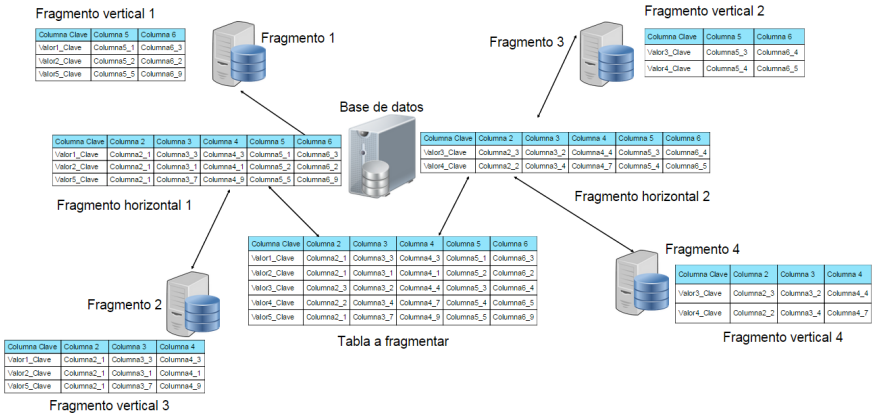

Figura 5.4: Fragmentación mixta

La fragmentación hibrida o mixta es aplicada cuando la fragmentación vertical le sigue un proceso horizontal, a esto nos referimos a que cuando los procesos horizontales se fragmentan los procesos verticales que faltan son fragmentados de forma vertical.

5.5. Aspectos a tomar en cuenta

Existen cuatro razones a tomar en cuenta cuando se piensa realizar cualquier tipo de fragmentación:

- La fragmentación es útil ya que aplicaciones de bases de datos pueden funcionar con vistas y por ello pueden ser utilizadas en distintas relaciones en distintos nodos.
- Se consigue una mejor eficiencia ya que los datos suelen estar almacenados cerca del nodo que más utiliza los datos.

- Permite aumentar el grado de concurrencia porque la fragmentación de las relaciones permite que una transacción pueda dividirse en sub-consultas que operan sobre estos fragmentos.
- Provee mayor seguridad, ya que los datos que no son utilizados por un nodo local no son almacenados en él y no pueden estar al alcance de personas que no tienen autorización a él.

5.6. Ventajas y Desventajas

- Uso: en general, las aplicaciones trabajan con vistas en vez de relaciones enteras. Por lo tanto para la distribución de datos es apropiado para trabajar con subconjuntos de relaciones como la unidad de distribución.
- Eficiencia: los datos son almacenados cerca de donde son frecuentemente más usados.
- Paralelismo: con los fragmentos como unidad de distribución, una transacción puede ser dividida en varias sub consultas que estén operando en fragmentos, esto debería de aumentar el grado de concurrencia y paralelismo en el sistema.
- Seguridad: los datos que son requeridos por aplicaciones locales no son almacenados y por consecuencia estos datos no están disponibles para usuarios no autorizados.
- Rendimiento: el rendimiento de las aplicaciones globales que requieren datos de varios fragmentos localizados en diferentes sitios podría ser lento.
- Integridad: el control de integridad puede ser más difícil si los datos y las dependencias funcionales son fragmentados y alojados en diferentes sitios.
- Mantenimiento: es difícil mantener la consistencia de datos a través de los sitios.

Capítulo 6

Administración para el servidor Apache Tomcat 7.0

6.1. Planteamiento de la propuesta

En la figura 6.1, se puede apreciar el esquema general de la propuesta planteada en esta tesis. La funcionalidad del sistema a desarrollar está dada por los siguientes pasos:

1. Contar con un servidor principal de aplicaciones web, en este caso el Apache Tomcat 7.0, un proceso monitor asociado, una o más máquinas participantes, también llamadas servidores secundarios, y n clientes que envían solicitudes HTTP al servidor principal, mismos que se pondrán en espera de recibir sus respectivas respuestas HTTP.

2. Cada que una máquina participante se registra, esta envía una solicitud y será el proceso monitor asociado el que le asigne trabajo y le proporcione la información que necesite. Con eso se evitarán los cuellos de botella o fallas en el servidor principal. En cualquier momento, una máquina participante puede pedir ser dada de baja.

3. Suponiendo que uno o más clientes envían un número considerable de varios tipos de peticiones HTTP, estas serán recibidas directamente por el servidor principal quien las reenviará a su proceso monitor asociado, siendo este último que tendrá a cargo las labores de balanceo y distribución de carga de trabajo, así como de replicación y fragmentación de información entre las máquinas participantes, mediante un mecanismo de comunicación TCP/IP de tipo P2P.

4. Para el manejo de la información se contará con una base de datos completa en el servidor principal y parcial entre las máquinas participantes.

5. Cuando a una máquina participante se le asigna una petición a ser atendida, esta al momento de atenderla va a reenviar al proceso monitor la respectiva respuesta, esta

respuesta el proceso monitor la propaga hacia el servidor principal, quien finalmente, la reenvía al cliente respectivo. Con esto se verifica la funcionalidad de la solución propuesta.

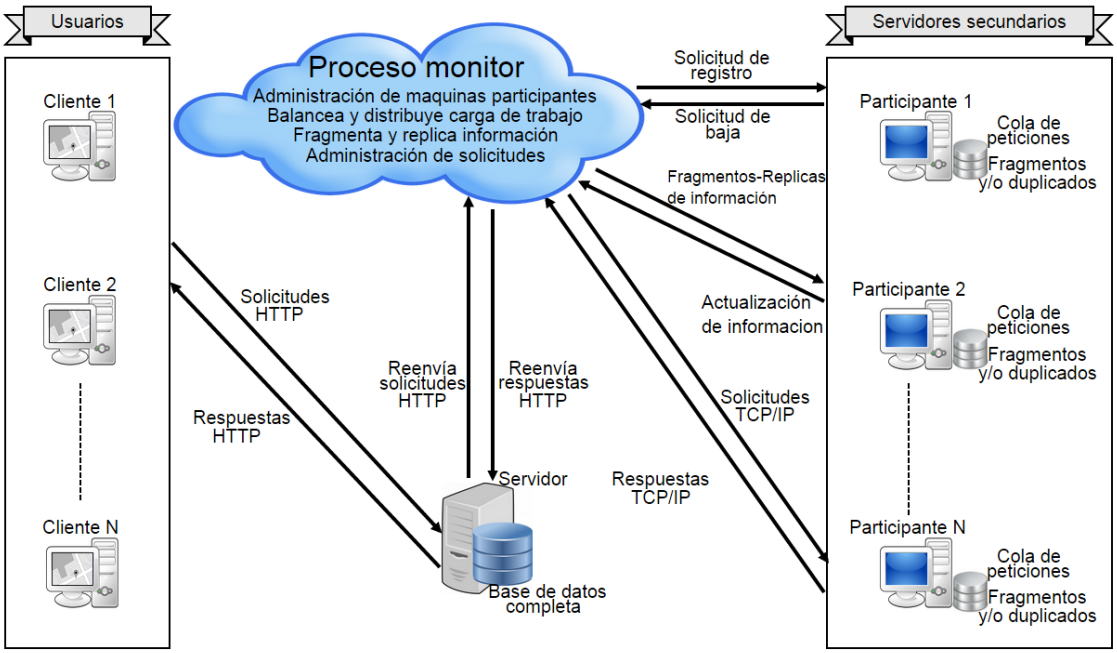

Figura 6.1: Esquema general de funcionalidad de la propuesta a implementar

7.3.6

Otra manera de apreciar la funcionalidad del esquema propuesto es la siguiente:

Cuando n usuarios están realizando M solicitudes de forma concurrente, el servidor principal recibirá las solicitudes y las pasará al proceso monitor, quien las encolará en una lista L conforme van llegando. De la lista L, el proceso monitor irá tomando la siguiente solicitud a ser asignada, junto con la información necesaria, a la máquina participante con menos carga de trabajo al momento, o en su defecto a cualquiera de las que tengan la carga más mínima, con ello se busca balancear y distribuir de manera más equitativa la carga de trabajo entre las máquinas participantes, así como el mantener replicas y fragmentos de información entre ellas. Toda respuesta generada por los servidores secundarios es enviada al proceso monitor y este la reenvía al servidor principal. Por último, toda respuesta recibida por el servidor principal será entregada al usuario que le corresponda.

6.2. Especificación formal mediante UML

Para efectos de introducir un modelado formal mediante UML de la propuesta introducida anteriormente, en esta sección se presentan y explican una serie de diagramas de secuencia que muestran las interacciones existentes entre los distintos actores, módulos o componentes del sistema a desarrollar. Estos diagramas son los mostrados a continuación:

6.2.1. Interacción usuario-máquina participante

La figura 6.2 dice que cuando el usuario ejecuta la aplicación, se le muestra de regreso una interfaz gráfica, en esta interfaz gráfica desplegará toda información de actividad en la que se encuentre involucrada la máquina. Así mismo, desde la interfaz misma el usuario podrá solicitar en cualquier momento cerrar la aplicación, a lo cual, se generará una solicitud de cierre, esta solicitud es validada para saber si el usuario confirma el cierre o declina el hacerlo, con lo cual la máquina cesa o continua con sus labores de manera respectiva. Todo recurso que sea administrado por la máquina será liberado al momento de su cierre.

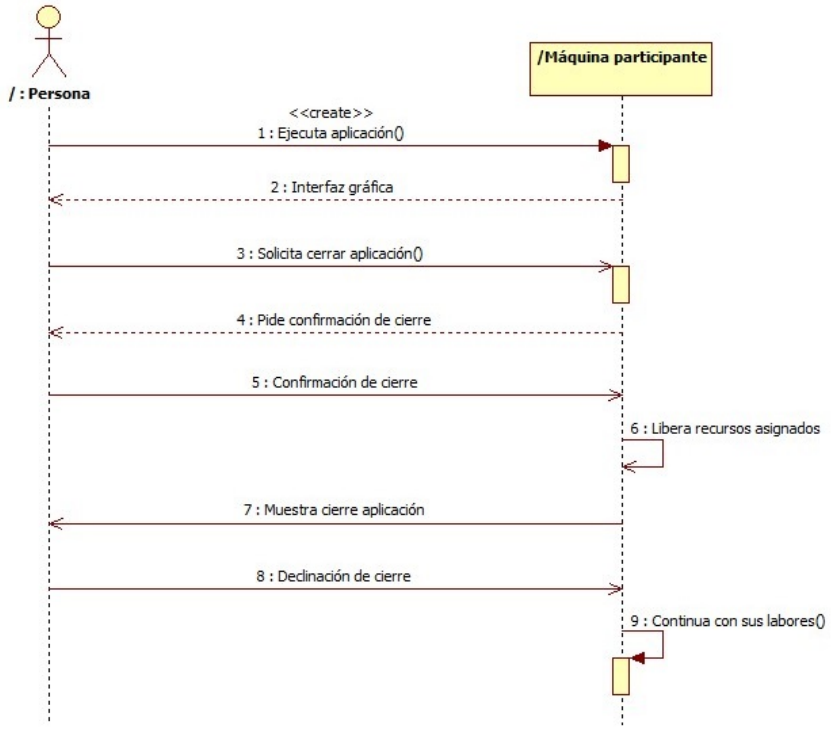

Figura 6.2: Persona máquina

6.2.2. Interacción servidor-proceso monitor

Esta interacción se muestra en la figura 6.3. Para que suceda dicha interacción, al inicio, el servidor debe registrarse en el monitor, mediante una solicitud de registro; el registro se hará en una tabla interna que manejará el monitor, aunado a ello se le informará al servidor que ya ha sido dado de alta; ahora bien, toda solicitud asíncrona HTTP que reciba el servidor por parte de un cliente será encolada y desencolada cuando le toque ser reenviada al monitor, el monitor por su parte va a balancear y distribuir la carga de trabajo, como a fragmentar y replicar la información entre las máquinas participantes para que estas realicen las tareas que les serán asignadas, cuando se recibe una respuesta el monitor verifica el destino de la respuesta, la reenvía y el servidor manda un mensaje confirmando que la respuesta fue enviada, creando un ciclo de reenvió de peticiones y respuestas entre el servidor y el monitor.

6.2.3. Interacción máquina participante-proceso monitor

En la figura 6.4, se muestra la secuencia que realiza el proceso monitor y la máquina participante, para que exista una relación entre estos dos, primero la máquina participante tiene que ser registrada en el monitor, mediante el envío de una solicitud de registro, dicho registro queda guardado en una tabla interna que maneja el monitor, el cual envía un mensaje de confirmación de registro a la máquina participante, el monitor inicia el encolamiento de las peticiones HTTP que son envidas por el servidor para que comience a balancear la carga de trabajo y así distribuir las peticiones entre las máquinas participantes registradas mediante solicitudes(mensajes) TCP/IP, las máquinas participantes reciben las solicitudes y se encargan de encolar, desencolar, generar las respuestas de las peticiones que les fueron asignadas y enviar las respuestas TCP/IP al monitor, el monitor al recibir cada una de las respuestas verificará a que máquina cliente le corresponde para que esta sea reenviada a su destino final mediante respuestas HTTP, si una máquina participante solicita darse de baja, esta es confirmada por el monitor y se hace una liberación de recursos.

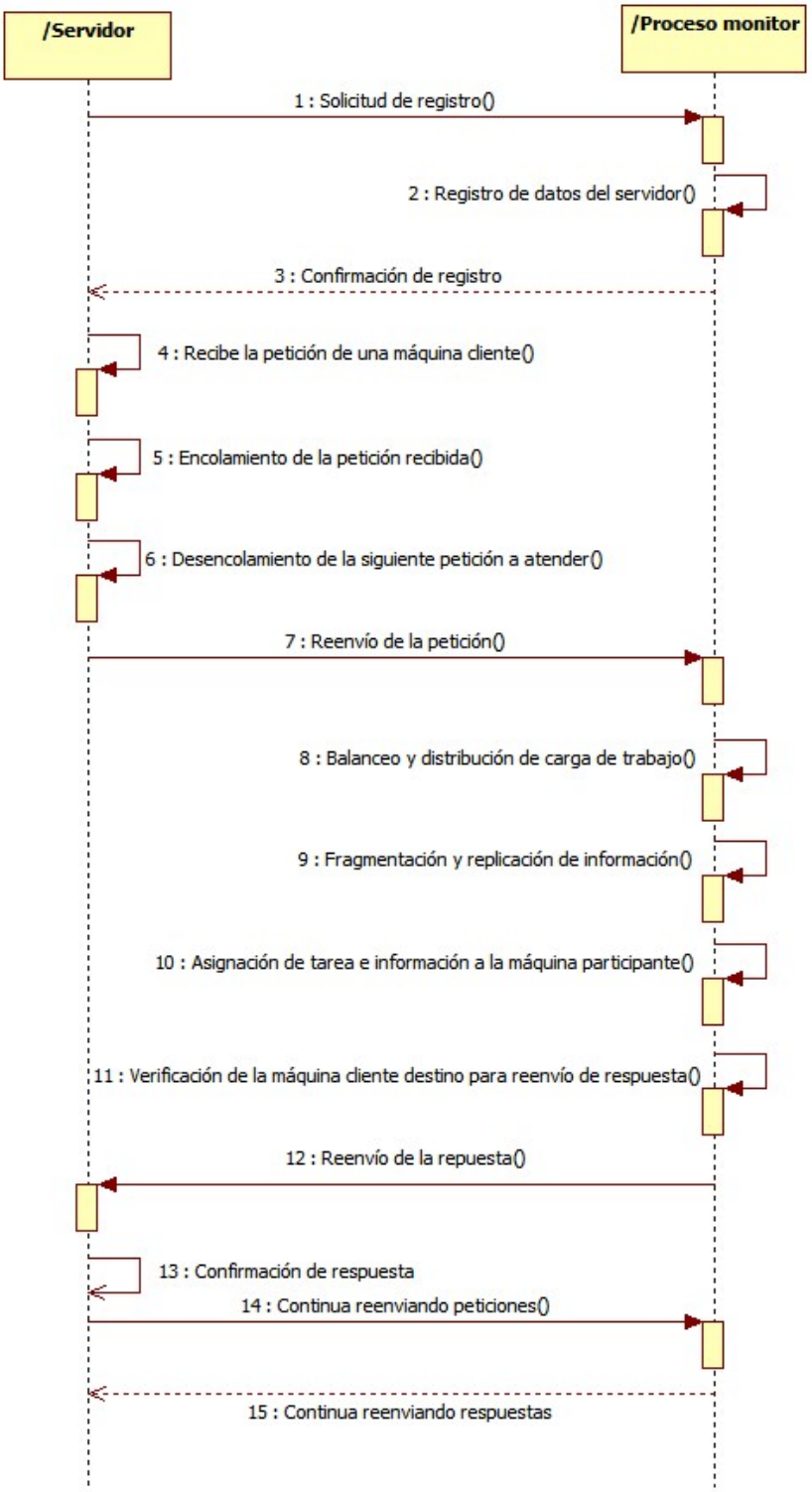

Figura 6.3: Servidor monitor

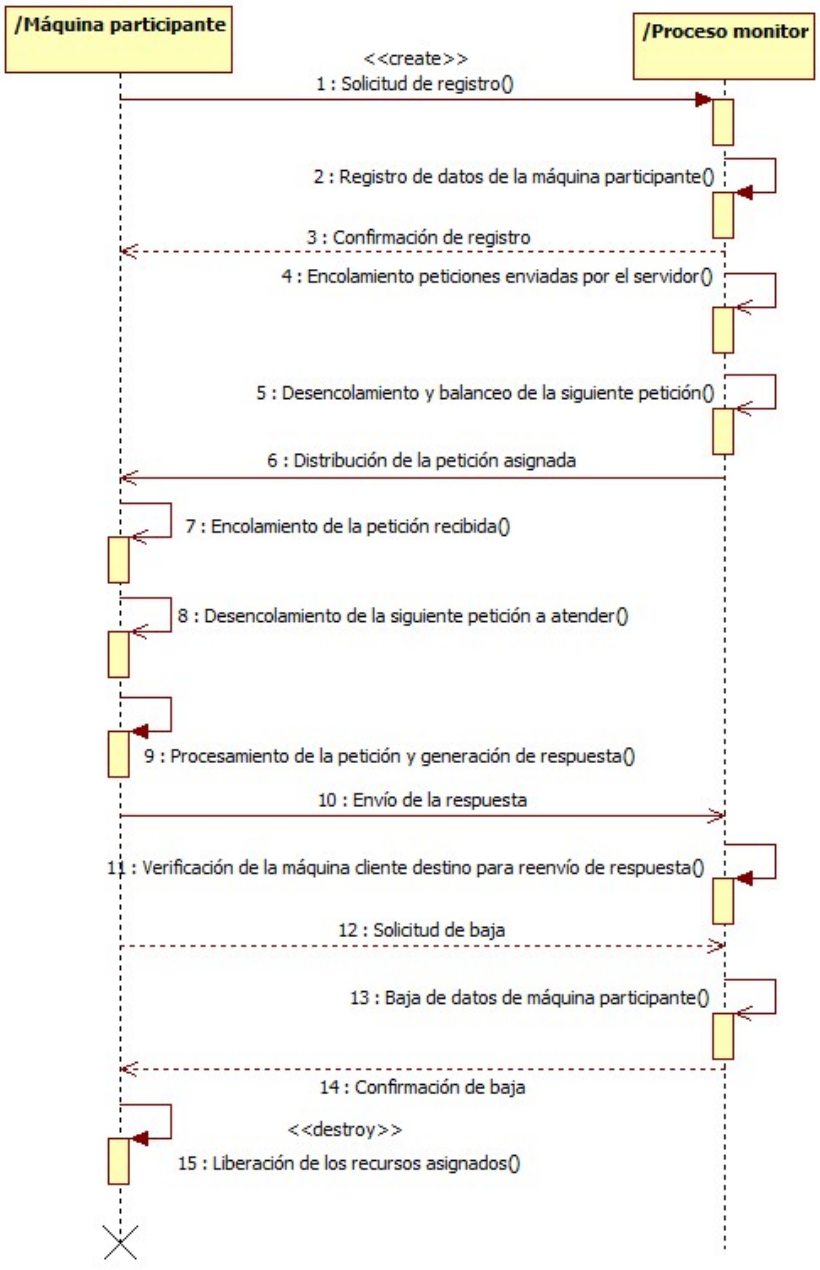

Figura 6.4: Máquina monitor

Para poder cubrir y cumplir con el resto de las secciones de este capítulo, se instaló y utilizó el servidor Apache Tomcat 7.0, el cual tiene funcionamiento de un contenedor de servlets, los cuales son las aplicaciones web. Además, se instaló J2EE [18] y J2SDK [38] los cuales serán utilizados para programar los servlets, mismos que serán ocupados en dicho servidor (ver Apéndice A).

6.3. Diagramas de clases

Hasta el momento, como parte del trabajo desarrollado para el cumplimiento de los objetivos planteados en este trabajo de tesis, se tiene una implementación parcial de la propuesta. Para tal efecto, se presentan los siguientes paquetes con sus respectivos diagramas de clases:

6.3.1. Paquete DB

En la figura 6.5 se puede apreciar a la clase DBManager, la cual contiene los componentes necesarios para tener acceso a una Base de Datos MySQL, como lo es el driver para realizar la conexión, así como la función de poder visualizar, insertar, eliminar y actualizar registros almacenados en la base de datos.

Por otra parte la clase DBOperation, esta encargada de realizar la creación de la base de datos cuando las máquinas participantes realizan la conexión con el monitor, así como las operaciones que realizaran los usuarios como lo son la alta y consulta de registros de reporte de tráfico.

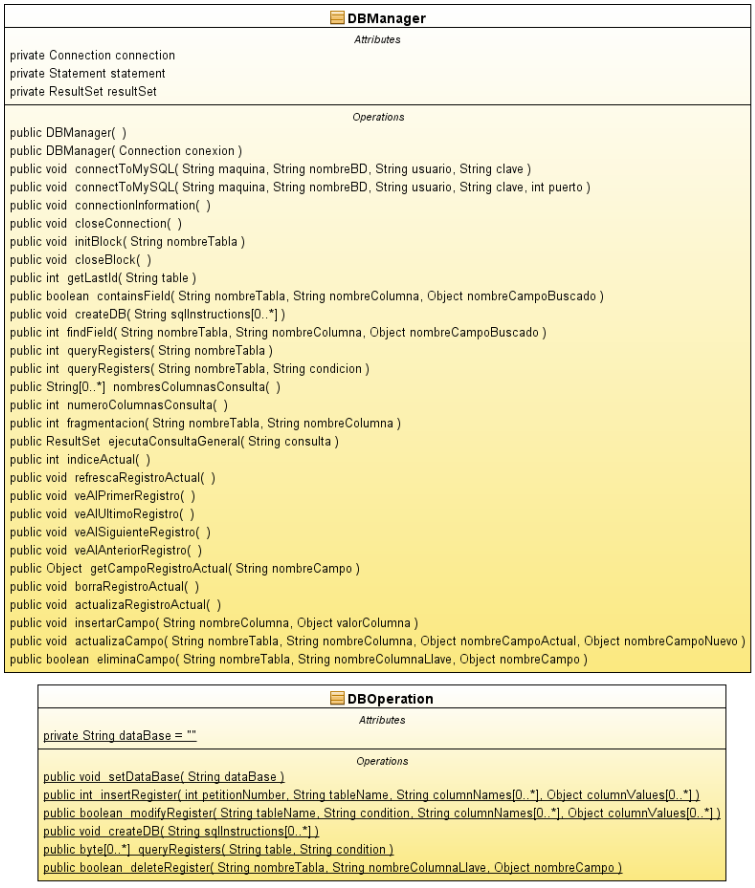

Figura 6.5: Diagramas de las clases DBManager y DBOperation

6.3.2. Paquete Module

En este paquete se crea la máquina participante la cual será la encargada de aceptar las peticiones que los usuarios realicen, así como la de crear la conexión a la base de datos y la creación de la misma para almacenar los registros de incidentes que se reciban y la conexión con el monitor; así mismo genera el modo gráfico de la máquina y muestra los mensajes de cada una de las peticiones que se hayan recibido y realizado (ver figura 6.6).

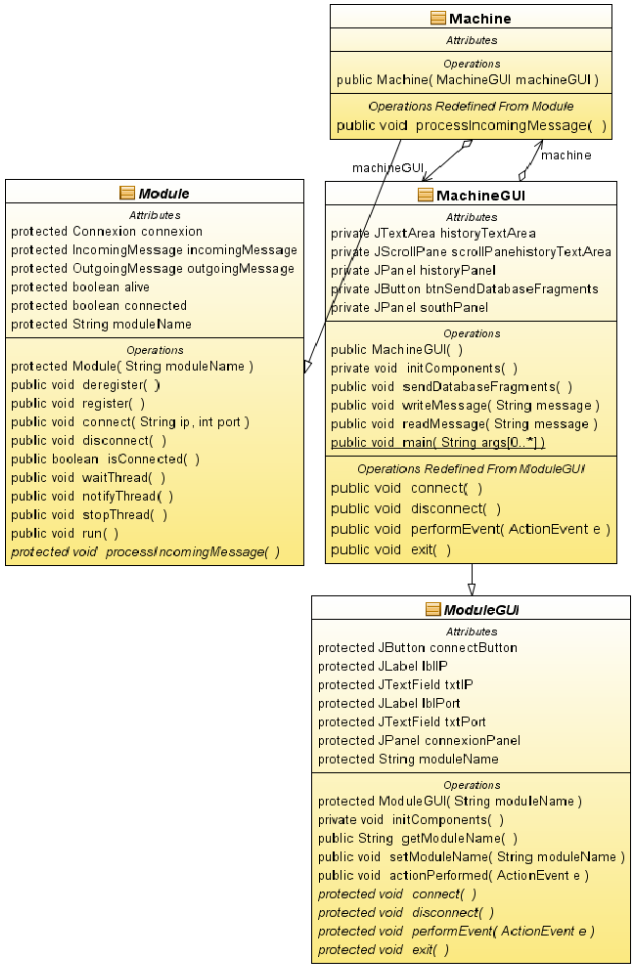

Figura 6.6: Diagramas de la relación de clases Machine-MachineGUI con Module-ModuleGUI

6.3.3. Paquete tcpCommunication

Como se puede apreciar en las figuras 6.7 , 6.8 y 6.9 este paquete realiza la interacción entre las máquinas participantes con el monitor, la aplicación web y la base de datos, y al mismo tiempo asigna las peticiones mediante el uso del código de paquete debido a que cada tipo de petición esta compuesta por un código de paquete, así la máquina participante sabrá que petición es la que se esta pidiendo atender y realizará la respuesta de estos por medio de mensajes creados en el mismo paquete.

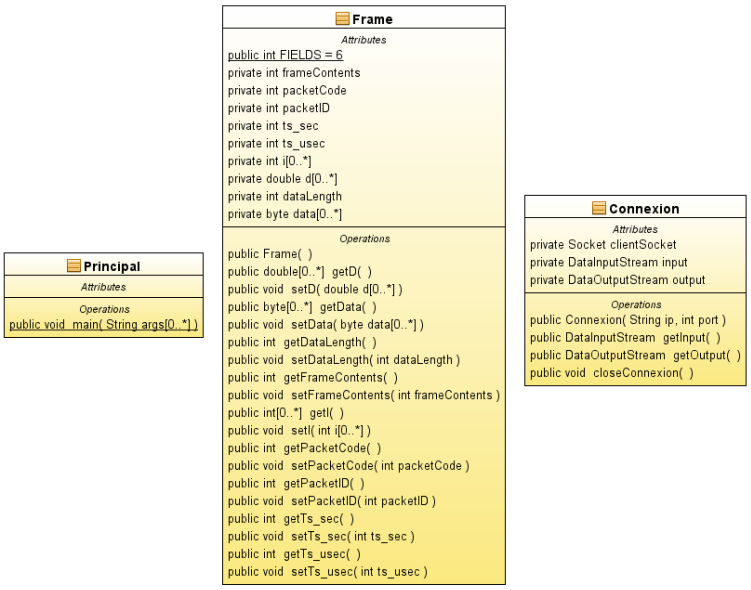

Figura 6.7: Diagramas de las clases Principal, Frame y Connexion

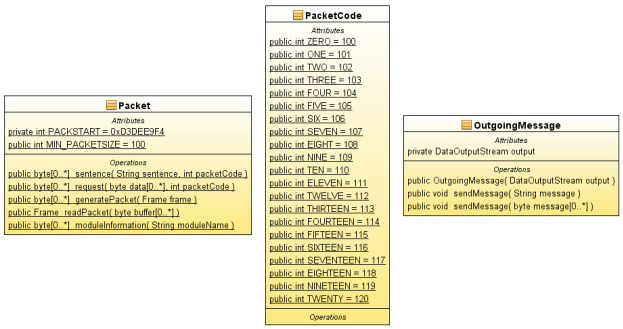

Figura 6.8: Diagramas de las clases Packet, PacketCode y OutgoingMessage

Figura 6.9: Diagramas de las clases IncomingMessage y PacketBuffer

6.3.4. Paquete Monitor

Realiza la interfaz gráfica del proceso monitor y se encarga de distribuir las peticiones equitativamente hacia las máquinas participantes para así enviar la información hacia la base de datos y generar la respuesta que sera enviada hacía el usuario, entre sus funciones de igual manera está el de crear un hilo el cual siempre está activo para recibir a las máquinas participantes y registrarlas cuando sean dadas de alta para atender las peticiones y el envío/recepción de mensajes hacia las máquinas (ver figuras 6.10 y 6.11).

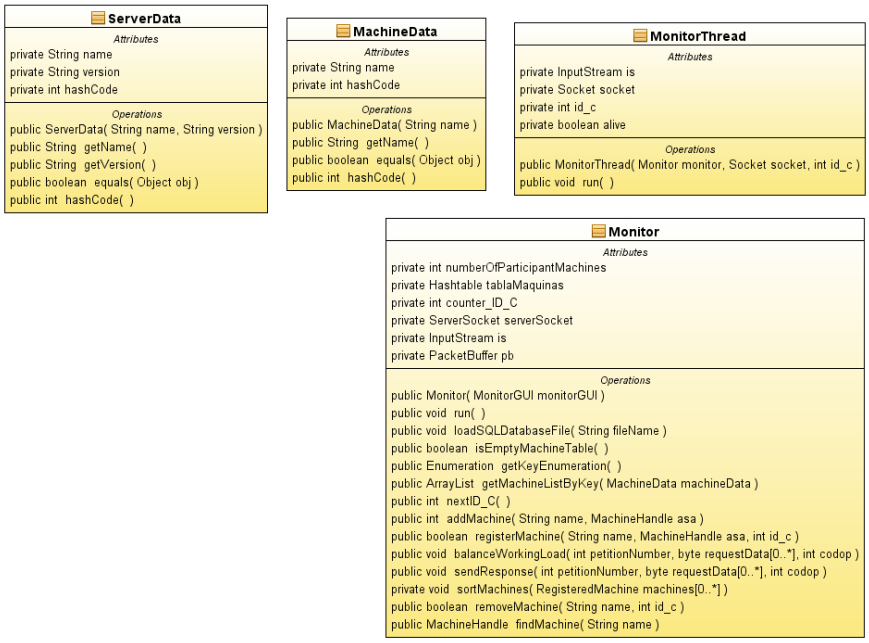

Figura 6.10: Diagramas de las clases MachineData y ServerData

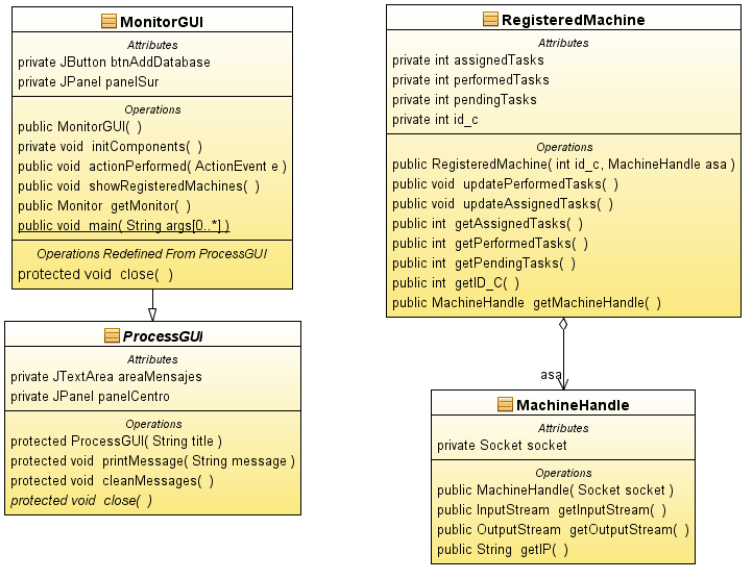

Figura 6.11: Diagramas de la relación de las clases MonitorThread y Monitor

6.3.5. Paquete servletUtilities

Este último realiza la conexión entre el monitor y la aplicación web, ya que es encargada de generar las vistas que podrá observar el usuario mediante las cuales podra generar las peticiones dentro de esta, como lo son la creación de reporte de incidentes y consulta de incidentes creados o registrados con anterioridad. El ServletThread crea el hilo que mantiene la comunicación activa entre estos dos, así como el ServletRequestResponse que se encarga de decodificar el código de paquete que genera la máquina participante cuando esta devuelve la respuesta que será mostrada finalmente al usuario. De igual manera genera las conexiones HTTP y el envío de las peticiones por medio de los paquetes (ver figura 6.12).

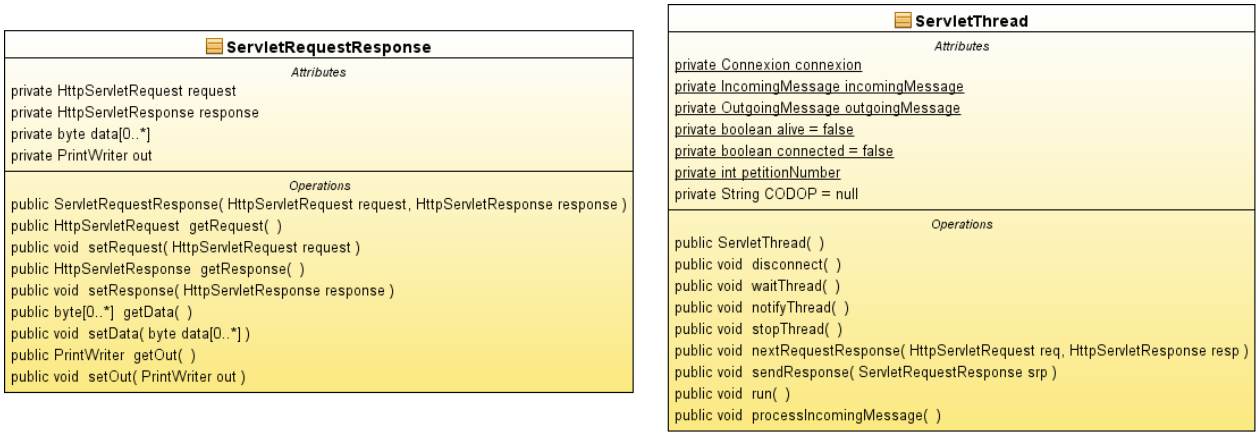

Figura 6.12: Diagramas de las clases ServletThread y ServletRequestResponse

6.4. Proceso monitor

El proceso monitor es la aplicación encargada de llevar a cabo dos funciones:

1. Balanceo y distribución de la carga de trabajo
2. Fragmentación y replicación de información

Ambas funciones son para que las máquinas participantes sepan qué peticiones deben atender y cuál información necesitan para ello.

Al estar activo esta en espera de aceptar n número de máquinas participantes las cuales fungirán como servidores secundarios que estarán encargados de agilizar la carga de trabajo cuando el número de peticiones sean mayores a las que el monitor sea capaz de resolver.

A continuación, en la figura 6.13 se muestra el monitor funcionando, el cual está libre de peticiones y de máquinas participantes.

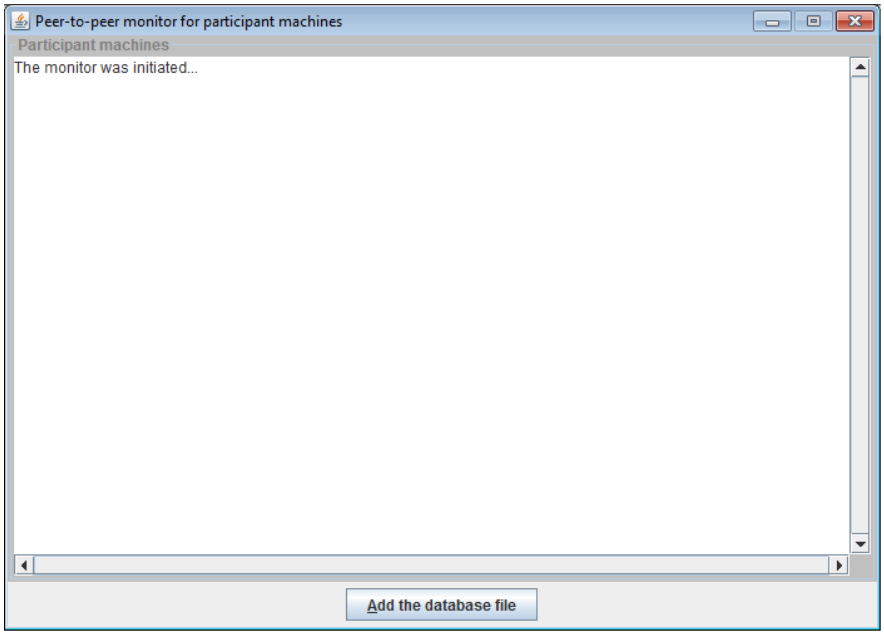

Figura 6.13: Monitor libre de máquinas participantes

Se tiene que dar de alta la base de datos para así poder dar inicio al uso de la aplicación mediante las máquinas participantes ya que serán las encargadas de crear la base de datos (ver figura 6.14).

Se observa que se tiene registrada en el monitor la máquina que fue dada de alta y ahora está lista para ser utilizada como un servidor secundario, mostrando de cada máquina su ID_C, Nombre, IP, puerto y número de tareas (asignadas, realizadas y pendientes) (ver figura 6.15).

Si es requerido se pueden dar de alta tantas máquinas como sean requeridas para contribuir al procesamiento de peticiones de los usuarios y estas sean realizadas en el menor intervalo de tiempo posible.

Cada vez que se inicialice una máquina esta será registrada en automático en el monitor (ver figura 6.16).

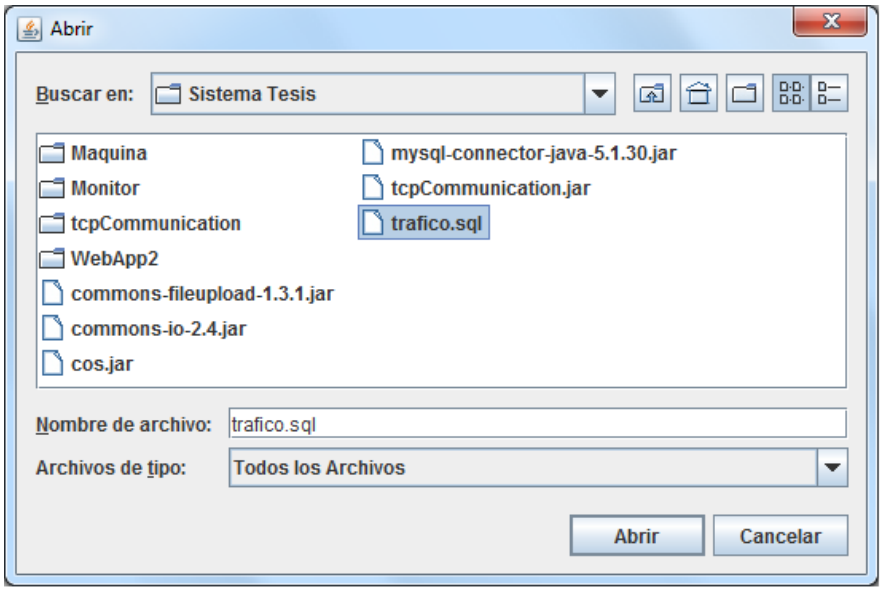

Figura 6.14: Selección de la base de datos, para permitir la conexión con las máquinas

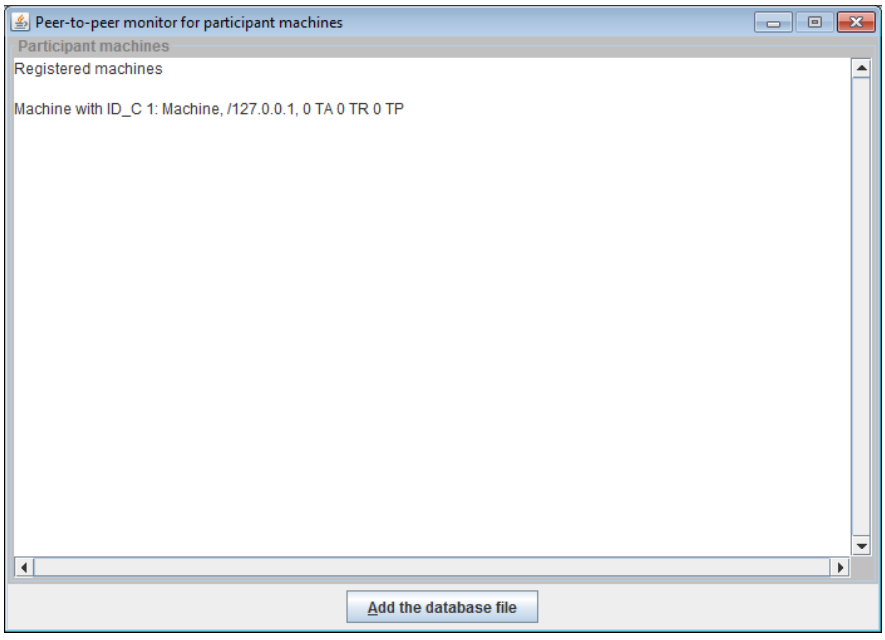

Figura 6.15: Monitor con una máquina registrada

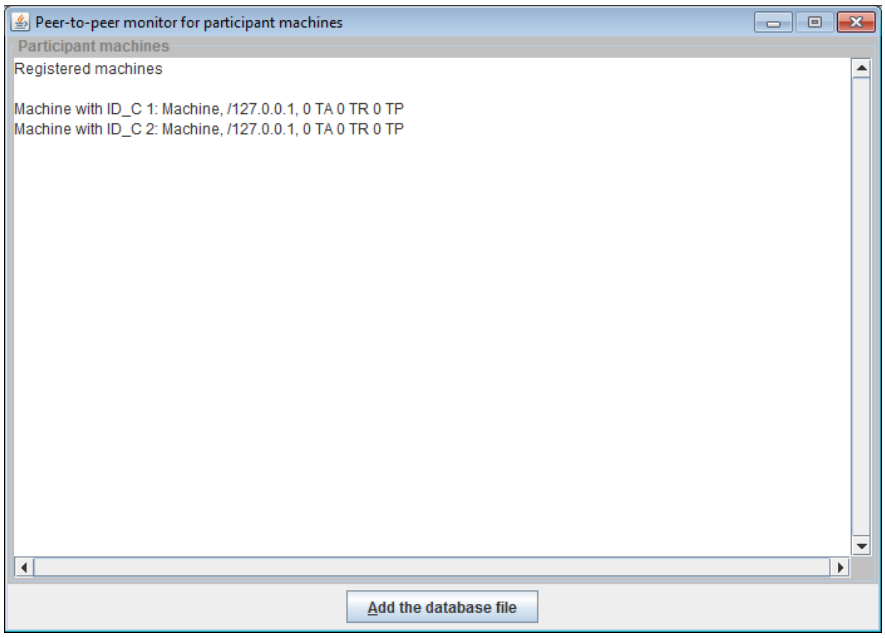

Figura 6.16: Monitor con dos máquinas registradas

6.5. Máquina participante

La máquina participante se encarga de liberar la carga de trabajo al proceso monitor, para el cual las peticiones serán repartidas a cuantas estén dadas de alta y registradas en el monitor. Almacena réplicas y fragmentos de información de las peticiones.

Esta es una máquina participante, la cual al ser dada de alta en automático es visible en el monitor (ver figura 6.17).

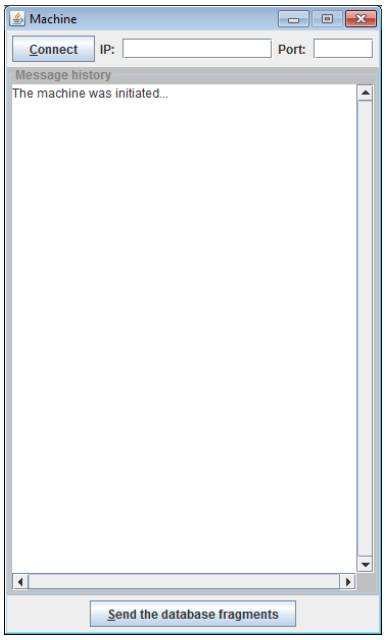

Figura 6.17: Máquina participante

Para que esta pueda funcionar correctamente se tiene que definir la dirección IP a la cual se conectará; en este caso debe de ser la dirección IP y el puerto que tenga el proceso monitor por el cual se conectará, debe estar dentro de la misma red de trabajo ya que hace uso del protocolo de red TCP/IP (ver figura 6.18).

Cuando una máquina participante es dada de alta está encargada de poder crear la base de datos para así realizar el proceso de fragmentación al finalizar las peticiones, para ser enviado al monitor y esta información sea almacenada, como se muestra en la figura 6.19.

Cada que una petición es atendida, la máquina participante registra el número de petición y esta las lista de tal manera que se pueda tener un registro de las peticiones que fueron atendidas por dicha máquina (ver figura 6.20).

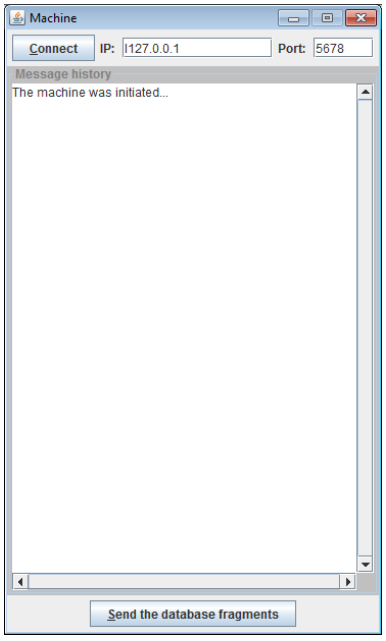

Figura 6.18: Definición de parámetros para conexión con el Monitor

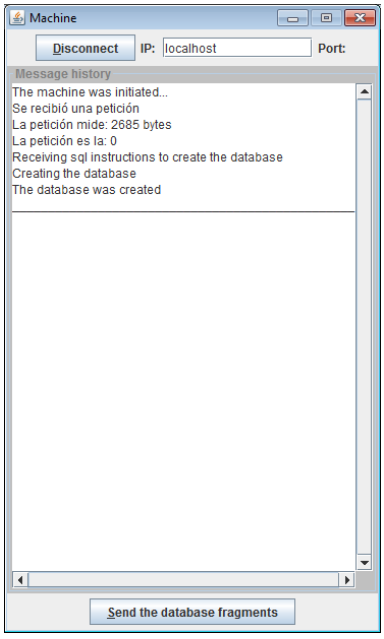

Figura 6.19: Máquina participante conectada

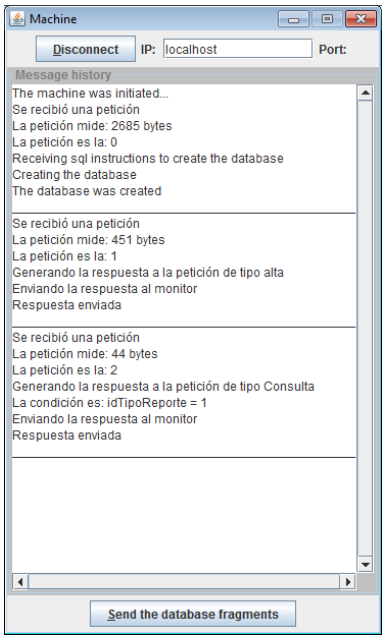

Figura 6.20: Máquina participante conectada

6.6. Caso de estudio

Para este tema de tesis referente al balanceo y distribución de carga de trabajo y replicación y fragmentación de información, se creó una base de datos la cual esta encargada de almacenar las peticiones de los usuarios y una aplicación web que permitirá al usuario poderse registrar, para así poder realizar la creación de incidentes de tráfico y consulta de los mismos.

6.6.1. Base de Datos

Se desarrolló una base de datos relacional, la cual consta de 4 tablas, las mismas que se llenan de forma automática cuando un usuario registrado da de alta un reporte de tráfico. El diseño de la base de datos se muestra en la figura 6.21:

Cada tabla tiene relación con otra tabla, la tabla principal es Usuario, de manera que esta tabla debe de tener registros de usuarios, ya que sin esto la tabla ReporteTráfico no se le podría agregar registros debido a que para poder registrar un incidente se pide la cuenta del usuario registrado. TipoReporte por otra parte se insertan los datos de forma automática al crearse la base de datos de tal manera que cuando se solicite registrar un incidente esta provea el tipo de reporte que se puede reportar en la aplicación web. Así finalmente la tabla

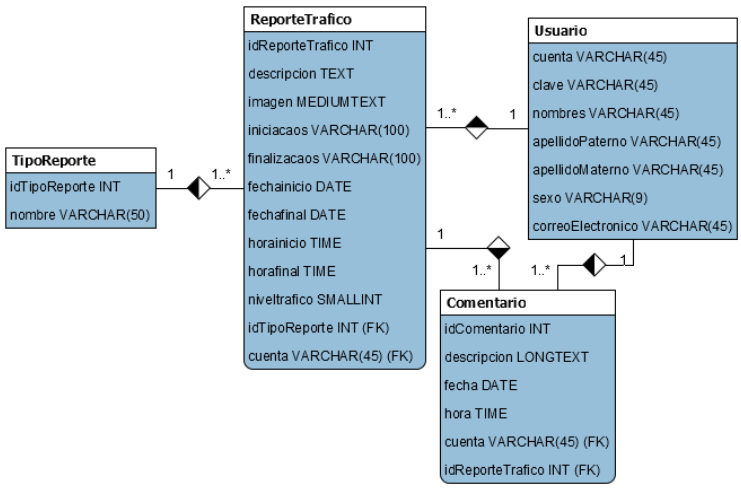

Figura 6.21: Página principal de la aplicación web

Comentario encargada de que algún usuario registrado pueda comentar un incidente de tráfico creado con anterioridad.

6.6.2. Aplicación web para reporte de tráfico vehicular

Se creó una aplicación web para el reporte y consulta de incidentes de tráfico vehicular (ver figura 6.22).

Como se puede observar en la figura 6.23, los incidentes de tráfico contemplados son los siguientes 11 tipos de reporte:

- Accidente: choques entre vehículos, atropellamientos,
- Auto Varado: por accidente, desperfecto mecánico
- Baches: saneamiento de carretera, producidos por condiciones climáticas
- Bloqueo: por razones climatológicas o de cierre por personas o vehículos
- Inundaciones: encharcamientos, desbordamientos de ríos, presas
- Manifestaciones: que afecten una o varias calles
- Obras: mantenimiento de calle, calles cerradas por carga o descarga de material peligroso

Figura 6.22: Página principal de la aplicación web

- Operativos: alcoholímetro, revisión a motonetas, revisión a carro particular
- Reten: revisión a unidades de transporte público o carga
- Semáforos: mala sincronización de tiempo, semáforos descompuestos, mala posición del semáforo
- Otros: riñas, balaceras

En la figura 6.24, se muestra que para poder reportar un incidente de tráfico vehicular, se requiere que todo usuario cumpla con lo siguiente:

Desde la cual primeramente como se muestra en la figura 6.25, se debe de dar de alta un usuario o acceder con uno que ya se encuentre registrado en el sistema, para así poder tener acceso a la aplicación web y realizar reportes sobre incidentes de tráfico.

Consecuente al registro o login en la aplicación se podrá acceder al sistema para así reportar en el mismo el tipo de incidente permitido dentro del catálogo dado de alta en el sistema, tal cual se aprecia en la figura 6.26.

Figura 6.23: Listado de incidentes contemplados

Figura 6.24: Registro de usuario para reportar incidentes

Figura 6.25: Página donde se realiza el login a la aplicación

Figura 6.26: Página principal del sistema de reportes de tráfico SRT-UAEMEX del Estado de México

Dentro de los reportes de tráfico permitidos por el sistema se encuentran los siguientes (ver figura 6.27):

Figura 6.27: Formulario de registro de incidentes de tráfico

Cuando se ha dado de alta o en caso de existir más de un reporte de trafico este será mostrado en la sección de reportes con solo seleccionar el tipo de reporte que se desee visualizar mostrando la información tal cual se muestra en la figura 6.28.

En caso contrario de que el reporte seleccionado no tenga ningún reporte de tráfico solamente se mostrará una pantalla como la de la siguiente figura 6.29 donde se menciona que no se encuentra dado de alta información sobre ese tipo de incidente.

6.6.3. Prueba realizada

Se solicitó la colaboración al grupo de 9° semestre de la carrera de Ingeniería en Computación, el G-91, para realizar un experimento en la Sala de Cómputo B del Edificio A del Centro Universitario UAEM Valle de México, el cual fue llevado a cabo el día 14 de septiembre de 2017 acudiendo 20 alumnos en total. El experimento fue llevado a cabo en cuatro fases, descritas como sigue:

6.6.3.1. Primera fase: levantamiento del servidor, monitor y máquinas participantes

- El servidor Apache Tomcat 7.0 fue levantado en la máquina con IP, en la cual se alojó la aplicación web “WebApp2”.

Figura 6.28: Página que muestra la información sobre los incidentes de tráfico

Figura 6.29: Página con resultado nulo de una búsqueda de incidentes

- Para el caso del monitor, este fue levantado en la máquina donde se ejecutó el servidor.
- Fueron levantadas 2 máquinas participantes, la primera en la máquina donde se ejecutó tanto monitor como servidor, la segunda en la máquina con IP: .
- Una vez levantadas las máquinas participantes estas se conectaron al monitor para recibir las instrucciones de creación de su respectiva base de datos, llamada "tráfico". Así cada máquina quedó lista para recibir y atender las peticiones y para manejar su propia información.

6.6.3.2. Segunda fase: registro de usuarios

- Cada alumno del grupo según indicaciones dadas ingresó a la aplicación "WebApp2"por medio de la dirección: http://172.17.242.42:8084/WebApp2/.
- Una vez cargada la página de inicio, se pidió a todos los alumnos ir al menú "Reportar"para crear su registro de usuario (ver figura 6.30) dando click al enlace "Regístrate aquí", el cual abrió el formulario de registro de usuarios que se muestra en la figura 6.31.

Figura 6.30: Página Reportar de la aplicación Web

- Desde el formulario una vez ingresados y enviados sus datos se les regresó a la página de reportar para que iniciaran sesión y se les mandará a la página mostrada en la figura 6.32 para quedar listos y así poder participar en paralelo en las fases siguientes.

Figura 6.31: Formulario de registro en la aplicación web

6.6.3.3. Tercera fase: reporte de un incidente de operativo policíaco

- Fue solicitado que de manera obligatoria se llenaran todos los campos del formulario antes de su envío.
- El lugar de inicio del operativo se fijó en la ubicación conocida como “El Puerto de Chivos”, localizada en el municipio de Nicolás Romero.
- De manera similar, se fijó un lugar de fin en la ubicación denominada “La Colmena”, localizada en el mencionado municipio.
- Dichos lugares, fueron ingresados eligiendo el respectivo botón de radio y auxiliándose de las coordenadas de tráfico ingresadas para localizar los lugares de inicio y fin desde el mapa mostrado soportado por Google Maps.
- Para la descripción se solicitó que se dijera que era un “Operativo vial”que estaba desde muy temprano, sin embargo, algunos alumnos la extendieron mostrando inconformidad por el incidente a reportar como afectados.
- La imagen que se pidió cargar, previamente fue copiada al escritorio de las máquinas de la Sala de Cómputo B, siendo distinta pero relacionada al incidente de tráfico vehicular que se pidió reportar.

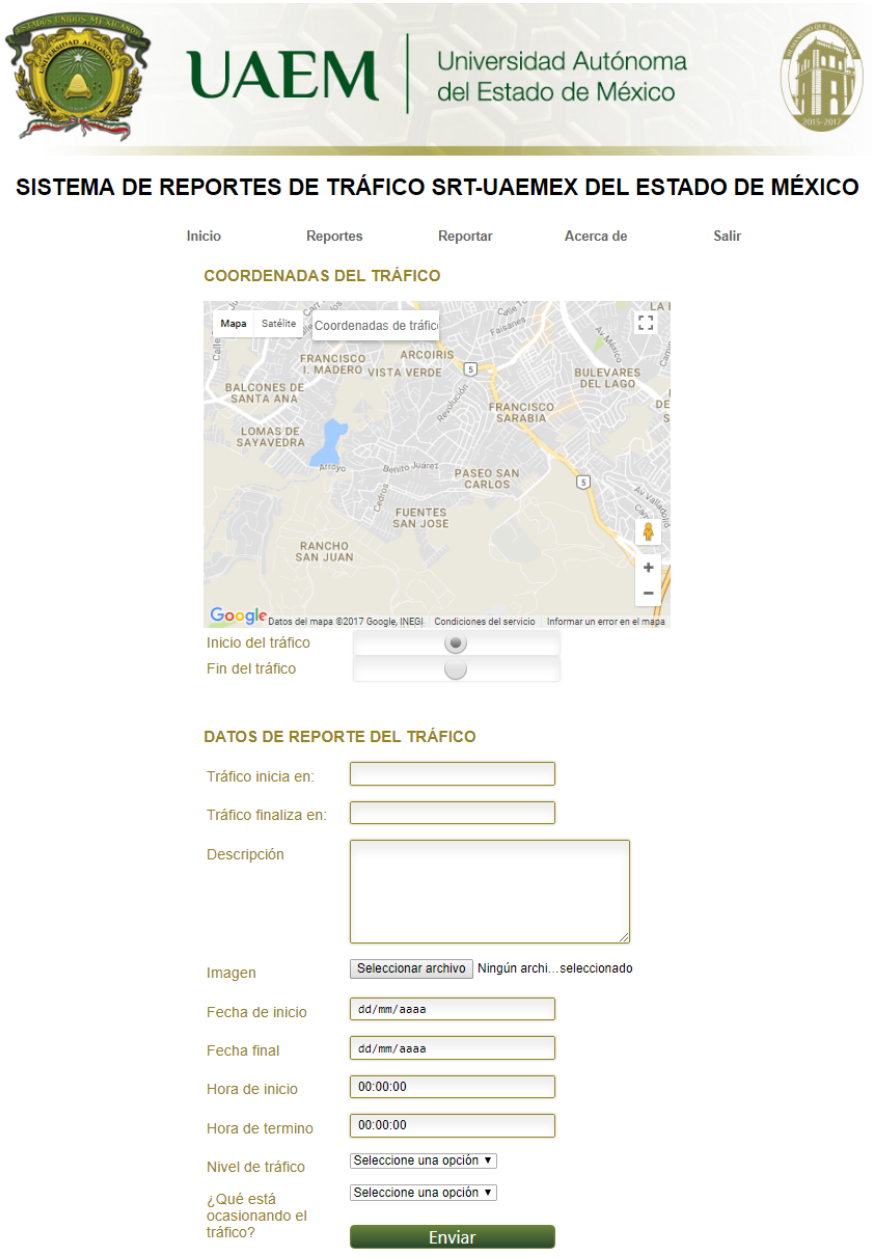

Figura 6.32: Formulario para reporte de incidentes vehiculares

- Todos fijaron las fechas de inicio y fin para el día “14/09/2017”.
- La hora de inicio fue dada para las “05:00:00”, y la de fin para las “18:00:00”horas.
- Se pidió que eligieran como nivel de tráfico al “Pesado”.
- Para completar el formulario, a todos se les indicó que tuvieran cuidado en indicar que lo que estaba ocasionado el tráfico fuera “Operativos”.
- Finalmente, se dio como orden que “a la cuenta de tres”todos debían enviar el formulario de reporte al mismo tiempo, es decir en paralelo, para que fuera posible apreciar el mecanismo de funcionamiento de la propuesta, es decir, el balanceo-distribución de carga de trabajo y fragmentación-replicación de información, para que fuera posible atender todas las peticiones de reporte, de tipo alta, en el menor tiempo posible por las 2 máquinas participantes. Este paso llevó a cada usuario a la página que se muestra en la figura 6.33, la cual indicó que fue atendida su petición de reporte como una solicitud de alta.

Figura 6.33: Página con resultado exitoso de la alta del reporte de incidente de tráfico

6.6.3.4. Cuarta fase: consulta de los reportes de incidentes de tráfico vehicular

- Una vez que fueron procesadas todas las peticiones de los alumnos y recibieron su respectiva respuesta, se les solicitó ir al menú “Reportes”, mismo que los mandó a la página que se muestra en la figura 6.34.

- Luego, se les indicó que seleccionaran el tipo de reporte de tráfico que correspondía al que habían mandado en la fase anterior, es decir, "operativos".
- Se les mencionó nuevamente que "a la cuenta de tres"todos debían de dar clic al botón de consultar, esto para que se pudiera apreciar nuevamente el mecanismo de funcionamiento de la propuesta y fuera posible atender todas las peticiones de tipo consulta, en el menor tiempo posible por las 2 máquinas participantes. En este paso llevó a cada usuario a la página que se muestra en la figura 6.35, la cual mostró todos los reportes de incidente de tráfico "operativos"que habían sido registrados como una solicitud de alta en la fase anterior. Los cuales fueron 20 en total, es decir, una por cada alumno.
- Ya que cada alumno pudo apreciar su respectivo reporte, se les pidió cerrar sesión yendo al menú "Salir"lo que provocó que su sesión finalizará. Cabe mencionar que se les agradeció por el apoyo brindado para poder llevar a cabo el experimento.

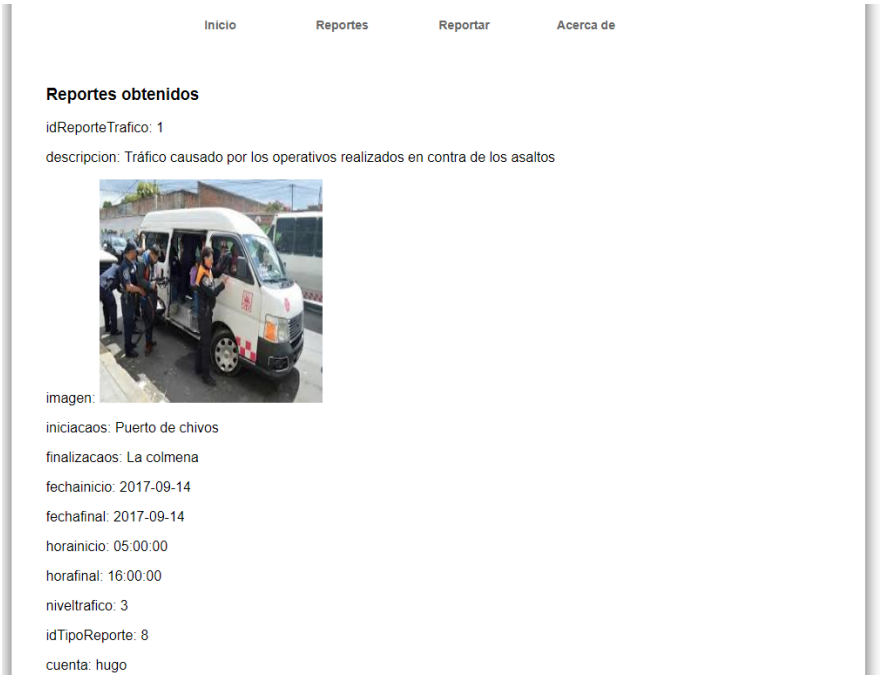

Figura 6.34: Página de consulta de reportes de incidentes

6.6.3.5. Quinta fase: envío de los fragmentos de la base de datos de cada máquina participante al monitor para una única y completa base de datos en el servidor

- En esta fase, no se requirió del apoyo de los alumnos.

Figura 6.35: Página de reportes

- Al término del experimento, desde la ventana de interfaz de cada máquina participante se dio click en el botón "Send the database fragments"para que la información almacenada en la base de datos de cada máquina participante, misma que se recopiló en la tercera fase, fuera enviada al monitor para que se almacenará en la base de datos del servidor y así tenerla completa.

6.6.4. Discusión de resultados

El propósito fundamental de este caso de estudio fue describir y verificar el funcionamiento del balanceo-carga de trabajo y de la distribución-fragmentación de información para el servidor Apache Tomcat 7.0 haciendo uso de una aplicación web como cama de pruebas o medio de validación, por medio del reporte y consulta de incidentes de tráfico vehicular a través de los usuarios registrados.

Como se explicó en las subsecciones previas, las pruebas realizadas para el caso de estudio se llevaron acabo en el laboratorio B del edificio A del Centro Universtario UAEM Valle de México, contando con el apoyo de 20 alumnos que fungieron como usuarios de la aplicación web, mismos que realizaron las peticiones de registro y consulta de incidentes de tráfico vehicular al mismo tiempo. Esto con el objetivo de corroborar que el proceso monitor cumpliera con su función de recibir las peticiones y asignarlas a las máquinas participantes mediante un balanceo de la carga de trabajo; por lo que debía primeramente determinar la máquina con menor carga de trabajo, y una vez ubicada poder asignarle la petición, logrando así agilizar la devolución de respuesta al usuario, reduciendo los tiempos de espera. En este caso de estudio, se contó con dos máquinas participantes, las cuales recibieron y procesaron sus respectivas peticiones asignadas según sigue a continuación:

1. Peticiones de alta
    - Máquina participante 1: procesó 4 peticiones de alta, siendo la más lenta. La figura 6.36 muestra los datos de las mismas.
    - Máquina participante 2: fue la que logró procesar el mayor número de peticiones, siendo 16 en total, tal como se puede apreciar en la figura 6.37 en donde se muestran las primeras 6 que procesó.
2. Peticiones de consulta
    - Cada máquina participante entregó los datos de las respectivas peticiones de alta que almacenaron, así los usuarios recibieron los correspondientes a las 20 peticiones que fueron hechas en total.

Además, para que las máquinas pudieran llevar a cabo sus funciones, contaron con su respectiva base de datos para el almacenamiento de la información que manejaron, lo cual dio prueba del mecanismo de fragmentación. Al termino de las pruebas, ambas maquinas participantes enviaron de regreso al proceso monitor los datos de reportes de alta de incidentes registrados para que fueran condensados en la base de datos del servidor, con lo que se observó la funcionalidad del mecanismo de replicación. Al final del caso de estudio se tuvo la información en una única base de datos (ver figura 6.38).

idReporteTrafico	descripcion	iniciacaos	finalizacaos	fechainicio	fechafinal	horainicio	horafinal	niveltrafico	idTipoReporte
1	operativo vial desde muy temprano. otra vez trafico pesado	Relleno Sanitario Puerto de Chivos, Ciudad Lpez Mateos, Mxico	la colmena, nicolas romero	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
2	Operativo vial desde muy temprano. Por culpa de esto llegue tarde a la escuela	El puerto de chivos, Nicolas Romero	La colmena, Nicolas Romero	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
3	Operativo vial desde muy temprano NADA DE AVANZAR! EL TRAFICO ES HORRIBLE! SIEMPRE ES LO MISMO!	El Puerto de Chivos, Nicolas Romero	La Colmena, Nicolas Romero	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
4	Carambola en el semaforo de bodegas. Mucha sangre	Bodegas de, Atizapan	El Pedregal, Atizapan	2017-09-15	2017-09-15	17:00:00	22:00:00	2	1

Figura 6.36: Datos de las peticiones de alta de incidentes de tráfico vehicular en la máquina participante 1

idReporteTrafico	descripcion	iniciacaos	finalizacaos	fechainicio	fechafinal	horainicio	horafinal	niveltrafico	idTipoReporte
5	Operativo vial desde muy temprano. Voy a llegar tarde otravez!	puerto de chivos, Nicolas Romero	La Colmena, Nicolas Romero	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
6	operativo vial desde muy temprano. por su culpa no llegue temprano a la esc	el puerto de chivos, nicolas romero	La Colmena, Francisco Sarabia, Villa Nicolás Romero, México	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
7	Operativo vial desde muy temprano. muy mal servicio	puerto de chivos, Nicolas romero	La Colmena, Nicolas Romero	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
8	Operativo Vial desde muy temprano. Este tipo de operativos afecta a los ciudadanos y pueden perder su trabajo.	El Puerto de Chivos, Nicolas romero	La Colmena, Francisco Sarabia, Nicolás Romero, México	2017-09-15	2017-09-15	17:00:00	22:00:00	2	1
9	operativo vial, que caos hay!!! no puede ser	el puerto de chivos, Nicolás Romero, México	La Colmena, Francisco Sarabia, Villa Nicolás Romero, México	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
10	operativo desde muy temprano.otra vez llegare tarde por estas cuestiones.	El puerto de chivos, Nicolas Romero	La colmena, Nicolas Romero	2017-09-15	2017-09-15	17:00:00	22:00:00	2	1

Figura 6.37: Datos de las peticiones de alta de incidentes de tráfico vehicular en la máquina participante 2

idReporteTrafico	descripcion	iniciacaos	finalizacaos	fechainicio	fechafinal	horainicio	horafinal	niveltrafico	idTipoReporte
1	operativo desde muy temprano.otra vez lleg...	El puerto de chivos, Nicolas Romero	La colmena, Nicolas Romero	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
2	Operativo vial Desde muy temprano. No lleg...	Relleno Sanitario Puerto de Chivos, Ciudad L�p...	La Colmena, Francisco Sarabia, Villa Nicol�s Ro...	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
3	Operativo vial desde muy temprano. Por cul...	El puerto de chivos, Nicolas Romero	La colmena, Nicolas Romero	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
4	Operativo vial desde muy temprano. Voy a ll...	El Puerto de chivos, nicolas romero	la colmena, nicolas romero	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
5	Operativo vial desde muy temprano. Voy a ll...	puerto de chivos, Nicolas Romero	La Colmena, Nicolas Romero	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
6	operativo vial desde muy temprano. por su ...	el puerto de chivos, nicolas romero	La Colmena, Francisco Sarabia, Villa Nicol�s Ro...	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
7	Operativo vial desde muy temprano. muy m...	puerto de chivos, Nicolas romero	La Colmena, Nicolas Romero	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
8	Operativo Vial desde muy temprano. Este ti...	El Puerto de Chivos, Nicolas romero	La Colmena, Francisco Sarabia, Nicol�s Romero...	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
9	operativo vial, que caos hay!!! no puede ser	el puerto de chivos, Nicol�s Romero, M�xico	La Colmena, Francisco Sarabia, Villa Nicol�s Ro...	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
10	operativo desde muy temprano.otra vez lleg...	El puerto de chivos, Nicolas Romero	La colmena, Nicolas Romero	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
11	Operativo v�al desde muy temprano no pu...	El puerto de chivos, Nicolas Romero	la Colmena, Nicolas Romero	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
12	Operativo vial desde muy temprano, solo qu...	el puerto de chivos, nicolas romero	La Colmena, nicolas romero	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
13	operativo vial desde muy temprano.llego tar...	Relleno Sanitario Puerto de Chivos, Ciudad L�p...	La Colmena, Francisco Sarabia, Nicol�s Romero...	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
14	Operativo vial desde muy temprano. ���...	El puerto de chivos, Nicolas Romero	La Colmena, Francisco Sarabia, Nicol�s Romero...	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
15	Operativo vial Desde muy temprano. No lleg...	Relleno Sanitario Puerto de Chivos, Ciudad L�p...	La Colmena, Francisco Sarabia, Villa Nicol�s Ro...	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
16	operativo vial desde muy temprano, otra ve...	Relleno Sanitario Puerto de Chivos, Ciudad L�p...	la colmena, nicolas romero	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
17	Operativo vial desde muy temprano. Siempr...	Relleno Sanitario Puerto de Chivos, Nicolas romero	La Colmena, Francisco Sarabia, Nicol�s Romero...	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
18	Operativo vial desde muy temprano NADA ...	El Puerto de Chivos, Nicolas Romero	La Colmena, Nicolas Romero	2017-09-14	2017-09-14	05:00:00	18:00:00	3	8
19	Carambola en el semaforo de bodegas. Habi...	Bodegas de Atizapan	El Pedregal Atizapan	2017-09-15	2017-09-15	17:00:00	22:00:00	2	1
20	carambola en el semaforo de bodegas hubo ...	bodegas de atizapan	el pedregal, atizapan	2017-09-15	2017-09-15	17:00:00	22:00:00	2	1

Figura 6.38: Datos de todas las peticiones de alta de incidentes de tráfico vehicular condensadas en una única base de datos

Capítulo 7

Conclusiones y trabajo a futuro

En este trabajo de tesis, se presentaron mecanismos de funcionalidad encargados de realizar la distribución-balanceo de carga de trabajo y la fragmentación-replicación de información en el servidor de aplicaciones web Apache Tomcat 7.0. Estos mecanismos, estuvieron soportados por un proceso monitor que fue el encargado de gestionar peticiones y administrar información entre máquinas participantes, cuya utilidad fue apoyar en disminuir la carga de trabajo en el servidor para evitar cuellos de botellas y así agilizar los tiempos de respuesta ante peticiones de alta y consulta de reportes de incidentes de tráfico vehicular a través de una aplicación web.

De los resultados obtenidos en el caso de estudio, fue posible apreciar que con la ayuda de máquinas participantes se pudo reducir el tiempo de respuesta para las peticiones de los usuarios, las cuales fueron hechas desde la aplicación web. Así en el servidor Apache Tomcat 7.0 no se provocó un cuello de botella y cada máquina realizó las respectivas peticiones que le fueron asignadas, logrando con ello que se atendieran una mayor cantidad de ellas en un menor intervalo de tiempo.

Desde el proceso monitor, se manejó una tabla de peticiones que se usó para registrar cada petición y la máquina a la que le fue asignada, así mismo, se manejó una lista de estado de las máquinas participantes que fue empleada para administrar sus tareas asignadas, realizadas y pendientes. Con esta lista, el proceso monitor al momento de asignar una nueva petición, lo hizo para la máquina que tuviera menos tareas pendientes (menor carga de trabajo), logrando con ello que fueran repartidas de manera balanceada y distribuida entre las máquinas participantes, para que pudieran ser resueltas en un intervalo de tiempo menor.

Cada que una máquina participante se registró dentro del proceso monitor al momento de conectarse, el mismo envío la instrucciones de creación de la base de datos local que debía administrar, es decir, la base de datos fue usada para agregar y consultar los registros de las peticiones que le fueron asignadas por el proceso monitor.

El uso del protocolo TCP se vio reflejado con la creación del paquete tcpCommunication, el cual se presentó en la subsección 6.3.3. Este paquete se desarrolló para soportar de manera confiable bajo un esquema orientado a conexión y de comunicación basado en sockets TCP/IP una comunicación confiable entre el proceso monitor y las máquinas participantes. Para ello, se definió una estructura propia de cada tipo de mensaje que fue enviado por medio de paquetes, los cuales fueron codificados y decodificados acorde a su código de operación. Este paquete, fue además usado para cubrir el requisito de tener una arquitectura P2P entre el proceso monitor y las máquinas participantes, haciendo que fungieran como clientes y servidores según fuera el caso.

En el caso de estudio, se creó una aplicación web para solicitudes de reportes y consultas de incidentes de tráfico vehicular, hechas por usuarios previamente registrados. Esta aplicación web fue la encargada de recibir las peticiones que fueron asignadas a las máquinas participantes por el proceso monitor, siendo con ello la vía de verificación y validación de los mecanismos de distribución-balanceo de carga de trabajo y replicación-fragmentación de información para el servidor de aplicaciones web Apache Tomcat 7.0.

Al término del caso de estudio, desde cada máquina participante se enviaron al proceso monitor los fragmentos que habían sido almacenados en su respectiva base de datos, una vez reunidos todos este último los envío hacia el servidor para que se tuviera una única base de datos.

Como trabajo a futuro, se puede realizar una extensión de la aplicación web mostrada mediante el desarrollo de una aplicación móvil para cualquier sistema operativo existente, como los son los sistemas Android e IOS [39], así como aplicaciones de escritorio en sistema Windows y Linux [40], además de poder ser escalada para permitir realizar comentarios en los reportes de incidentes de tráfico vehicular y cerrarlos pasado un periodo de tiempo desde que se reportó por primera vez el incidente.

Finalmente, es necesario proveer un mecanismo de confiabilidad más robusto, mismo que garantice que toda petición pueda ser atendida y que en caso de que una máquina falle se pueda tener forma de recuperar la información que administra y de atender las peticiones que haya tenido en espera, o bien, la que hubiera estado en atención al momento de la falla. Este mecanismo de confiabilidad se puede hacer ya sea para restablecer la máquina participante que falle, o bien, para usar una máquina participante auxiliar que retome su trabajo en curso y pendiente, esta máquina auxiliar puede ser una nueva o alguna de las que estén en funcionamiento.

Apéndice A

Arquitectura J2EE

La arquitectura de la plataforma Java 2 Edición Empresarial (J2EE, por sus siglas en Inglés Java 2 Platform, Enterprise Edition) usa un modelo de aplicación distribuida multinivel para aplicaciones empresariales y aplicaciones basadas en la web. J2EE [41], permite utilizar arquitecturas de n capas distribuidas y se apoya ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones, esta arquitectura soporta una gran variedad de tipos de aplicaciones, como pueden ser desde aplicaciones web de gran escala a pequeñas aplicaciones cliente-servidor.

A.1. Introducción a la arquitectura J2EE

El objetivo principal de la arquitectura J2EE es crear un simple modelo de desarrollo para aplicaciones empresariales utilizando componentes basados en el modelo de aplicación y para esto J2EE consiste en cuatro elementos mayores:

- El esquema de aplicaciones J2EE es el modelo estándar de programación usado para facilitar el desarrollo multinivel.
- La plataforma J2EE incluye políticas necesarias y API's tales como los servlets y el servicio de mensajes java.
- La compatibilidad J2EE asegura que los productos son compatibles con las plataformas estándar.
- La referencia J2EE de implementación explica sus capacidades y proporciona su definición operativa.

La aplicación lógica está dividida en componentes de acuerdo a la función, los diversos componentes que forman una aplicación J2EE se instalan en diferentes máquinas dependiendo del nivel en el entorno J2EE de varios niveles a la que pertenece el componente de la aplicación.

Los componentes por los cuales están divido J2EE son los siguientes: cliente, web y de negocio, a lo que cada componente tiene una forma diferente de trabajar, en el componente cliente se trabajan con las aplicaciones del cliente a las cuales se refiere como páginas web dinámicas generadas con código HTML, XML entre otros más y applets generadas con código java, en el caso del componente web es donde se encuentran los servlets y páginas JSP, aquí los servlets se basan en procesar las peticiones y construir las respuestas, en cambio, las páginas JSP se encargan de crear contenido web estático y por último en el componente negocio es donde se ejecutan los enterprise javabeans que se encargan de resolver la lógica de negocio, siendo esto cuando se reciben los datos de los programas clientes, los procesa y los envía a la capa del sistema de información empresarial para ser almacenado.

A.2. Introducción al servidor de aplicaciones Apache Tomcat 7.0

Tomcat es un servidor de aplicaciones web de Apache Software Foundation que ejecuta servlets, HTML y java server pages (JSP) que devuelve páginas web. Descrito como una referencia de implementación de servlets y JSP, Tomcat es el resultado de una colaboración abierta de desarrolladores y está disponible en el sitio web apache. Apache Tomcat 7.0 proporciona por defecto un conector HTTP en el puerto 8080, también puede ser utilizado como servidor HTTP a pesar de que el rendimiento de Tomcat no sea tan bueno como el rendimiento del servidor apache HTTP.

A.2.1. Instalación y configuración

- Paso 1: para descargar Tomcat se tiene que acceder a su página oficial http://tomcat.apache.org y del lado izquierdo se observa la sección descargas (downloads) se elige una versión, en este caso se utilizará la versión 7.0 (ver figura A.1).

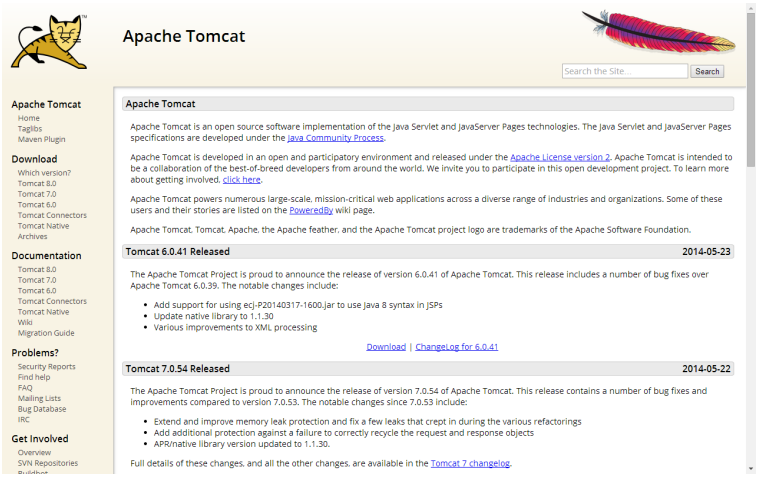

Figura A.1: Descarga del Apache Tomcat 7.0 desde la página web

- Paso 2: en esta parte se muestran dos opciones, de 32-bit/64-bit Windows Service Installer (pgp, md5), esta opción hace que se ejecute de forma automática el instalador, se le da clic para que inicie la descarga y al concluir la descarga se guardará en automático. Lo cual se ve en la figura A.2.

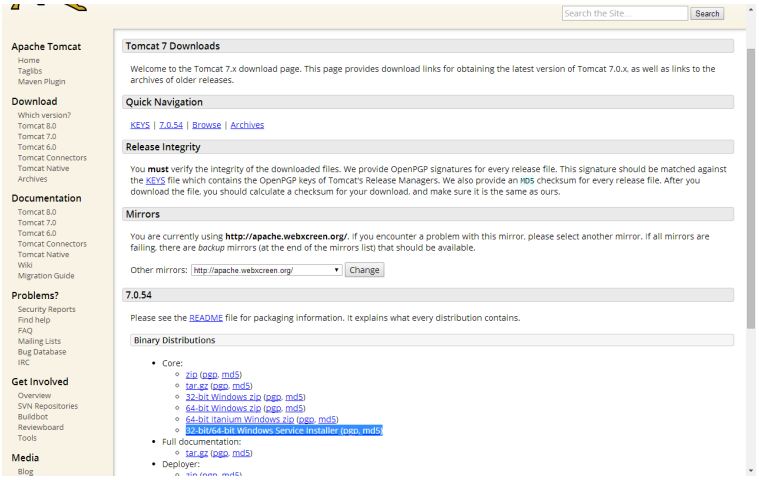

Figura A.2: Selección del instalador

- Paso 3: ya que haya terminado la descarga del archivo, se le dará doble clic para iniciar la instalación, la cual abrirá una ventana que muestra el inicio de la instalación, se selecciona aceptar para seguir con ella (ver figura A.3).

Figura A.3: Instalación de Apache Tomcat 7.0

- Paso 4: en la siguiente ventana muestra el acuerdo de licencia, se acepta dicho acuerdo para así poder proseguir con la instalación (ver figura A.4).

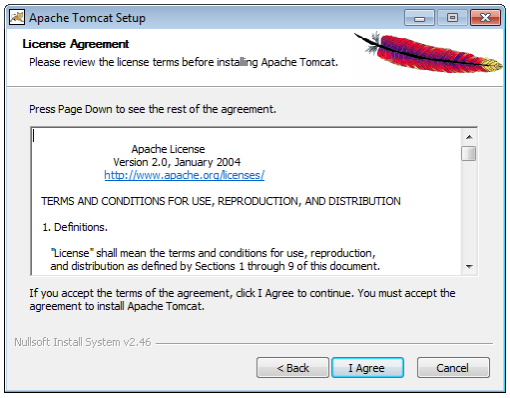

Figura A.4: Aceptar acuerdo de licencia

- Paso 5: como se muestra en la figura A.5 la ventana a continuación muestra los diversos tipos de instalación que contiene, en este caso será una instalación completa.

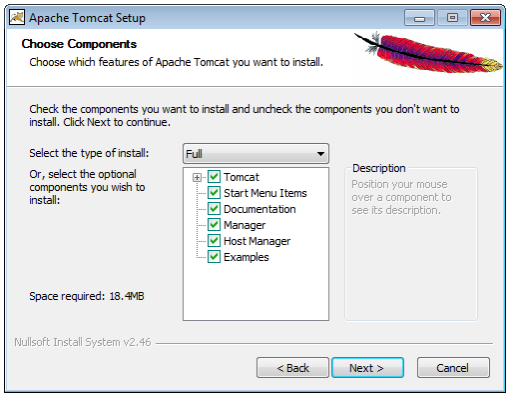

Figura A.5: Selección del tipo de instalación

- Paso 6: la siguiente ventana muestra la configuración del servidor Apache Tomcat 7.0, hay que ingresar los datos solicitados en este caso será usuario y contraseña, los puertos están configurados automáticamente por lo cual no habrá razón de moverlos, solo en caso de que alguna otra aplicación use el puerto que sugiere Apache Tomcat 7.0 para usar como se muestra en la figura A.6.

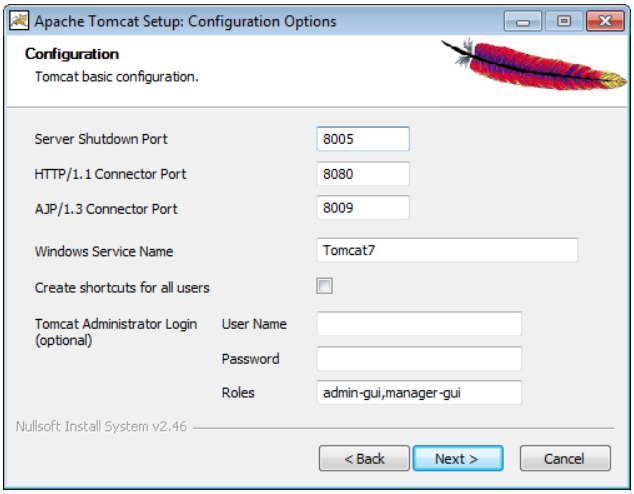

Figura A.6: Registro de usuario y contraseña en Apache Tomcat 7.0

- Paso 7: en la siguiente ventana muestra donde se encuentra ubicada la Máquina Virtual Java (JVM), esta máquina es de gran ayuda cuando se está compilando o trabajando con los archivos dentro de Apache Tomcat 7.0, se le da clic en siguiente (ver figura A.7).

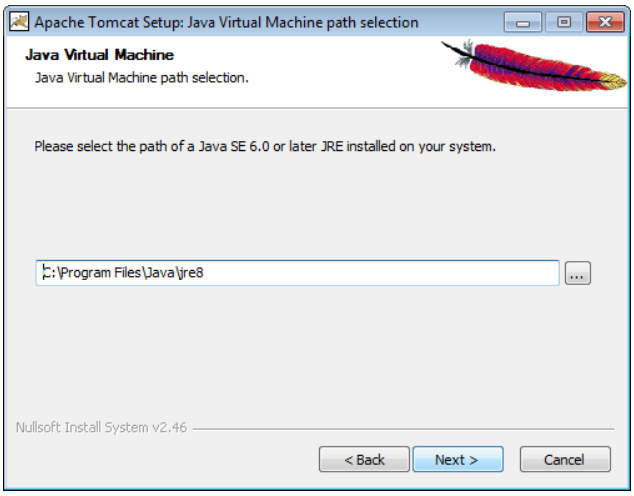

Figura A.7: Ubicación de la JVM (Java Virtual Machine)

- Paso 8: la siguiente ventana será la última, porque aquí muestra el lugar en donde se realizará la instalación, si la dirección es la deseada se da un clic en el botón instalar (ver figura A.8).

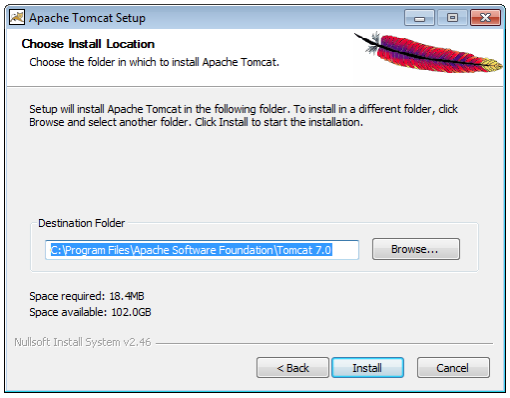

Figura A.8: Destino de la instalación

- Paso 9: como se aprecia en la figura A.9 tomará unos segundos en que termine de instalarse el servidor.

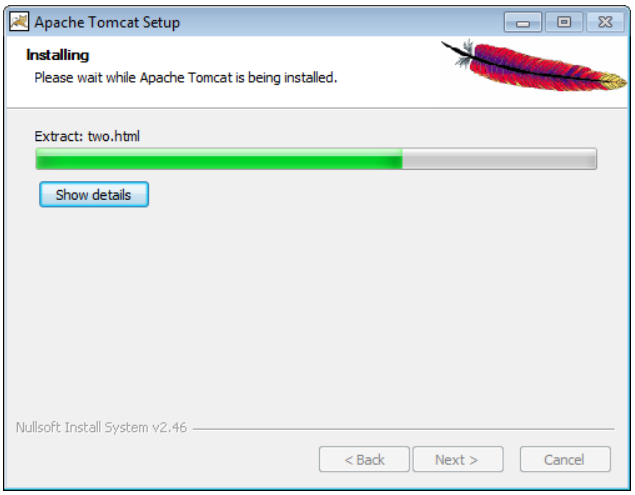

Figura A.9: Proceso de instalación

- Paso 10: mostrará una ventana la cual dará la opción de ejecutar el servidor, se da clic en finalizar y automáticamente arrancara el servidor (ver figura A.10).

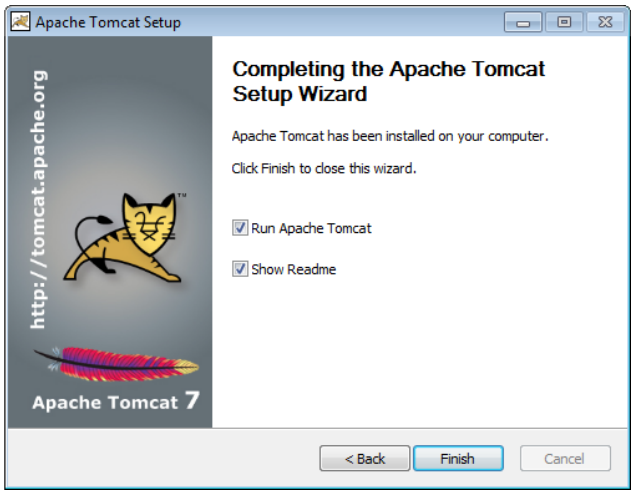

Figura A.10: Opciones al terminar la instalación

- Paso 11: este icono muestra que el servidor está funcionando correctamente y está activo para iniciar sesión sobre él (ver figura A.11).

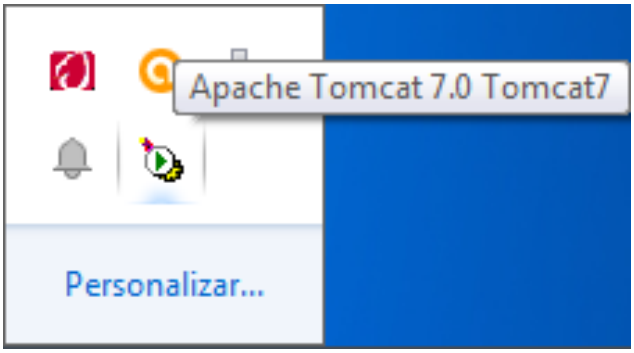

Figura A.11: Icono que muestra cuando Apache Tomcat 7.0 está activo

- Paso 12: solo queda abrir un navegador web, ir a la dirección localhost:8080 para que aparezca la página del servidor tomcat y se inicie sesión como lo demuestra la figura A.12.

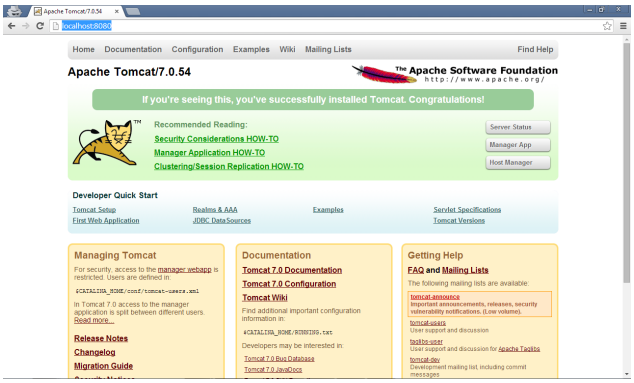

Figura A.12: Acceso al servidor Apache Tomcat 7.0

- Paso 13: por último se ingresa el usuario y contraseña para poder iniciar sesión y acceder a los servicios de Apache Tomcat 7.0 en el botón llamado Manager App (ver figura A.13).

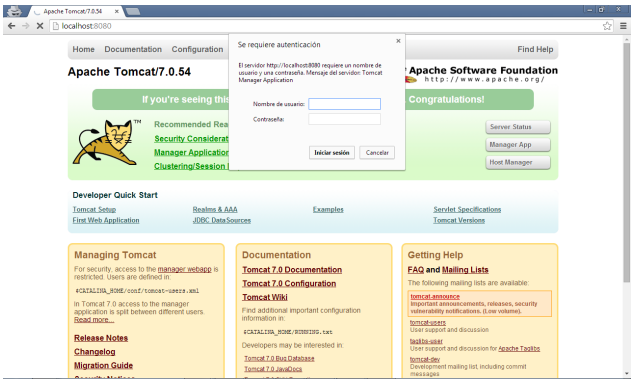

Figura A.13: Registro para acceder Apache Tomcat 7.0

- Paso 14: ambiente de trabajo en Apache Tomcat 7.0 (ver figura A.14).

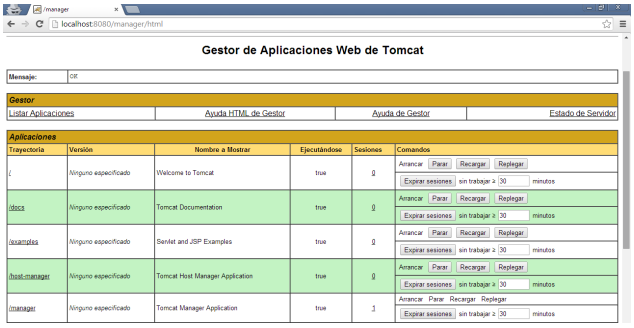

Figura A.14: Ambiente donde se darán de alta o de baja las aplicaciones WAR

A.2.2. Modelo del contenedor web

Como se puede apreciar en la siguiente figura A.15 se muestra la estructura de un contenedor web, la cual da el seguimiento que lleva las peticiones enviadas y las respuestas recibidas.

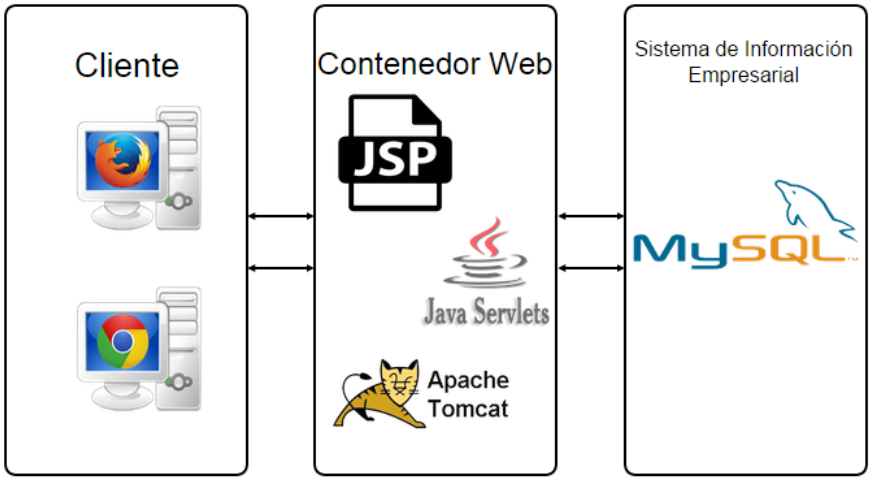

Figura A.15: Contenedor de aplicaciones web

A.2.2.1. Los archivos .WAR

Un archivo WAR (Web Archive) es una representación de una aplicación web en una única unidad distribuible. Es el método estándar empleado para empaquetar una aplicación web y dejarla lista para su distribución y acceso a través de servidores web con soporte para servlets y páginas JSP. No importa el número o tipo de recursos (Servlets, JSP, HTML, etc.), un archivo WAR agrupa una aplicación completa, en una única unidad de distribución, en un único archivo.

A.2.2.2. Estructura de una aplicación web

Las aplicaciones web constan de varias partes que se organizan en varios directorios

- Directorio raíz de la aplicación web
  - Archivos HTML, JSP, CSS, JS, imágenes, etc. que son visibles a los clientes de la aplicación.

- /WEB-INF/web.xml
  - Web Application Deployment Descriptor
  - Archivo XML que describe los servlets y otros componentes de la aplicación, además de parámetros de inicialización y restricciones de seguridad
- /WEB-INF/classes/
  - Clases java y recursos asociados: servlets y no servlets que no estén contenidos en ficheros JAR
- /WEB-INF/lib/
  - Archivos JAR: Librerías de clases, drivers JDBC, etc.
- En $CATALINA_HOME/lib
  - APIs Servlet 3.0 y JSP 2.1
  - XML Parser conforme a JAXP para procesar documentos XML

En la siguiente figura A.16 se puede ver la estructura de directorios de una aplicación web:

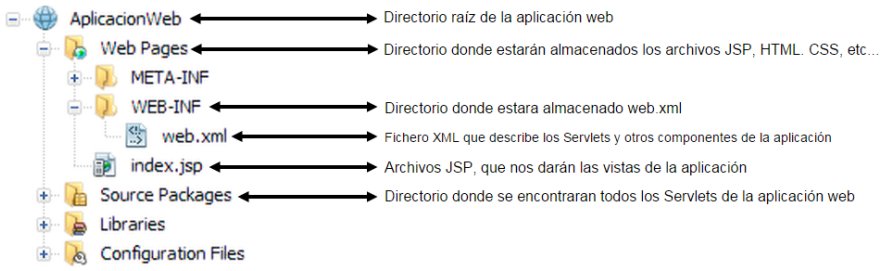

Figura A.16: Directorio de aplicación web

A.2.2.3. El descriptor de aplicaciones web: web.xml

Las aplicaciones web java utilizan un archivo de descriptor de implementación para determinar la forma en que las URL se asignan a los servlets y las direcciones URL que necesitan autenticación, entre otra información. Este archivo se llama web.xml y se encuentra en el WAR de la aplicación, concretamente, en el directorio WEB-INF/. web.xml forma parte del estándar de servlet para las aplicaciones web.

El descriptor de implementación de una aplicación web describe las clases, los recursos y la configuración de la aplicación, así como la forma en que el servidor web los utiliza para suministrar las solicitudes web. Cuando el servidor web recibe una solicitud para la aplicación, utiliza el descriptor de implementación para asignar la URL de la solicitud al código que debe controlar dicha solicitud.

El descriptor de implementación es un archivo denominado web.xml. Se encuentra en el WAR de la aplicación, concretamente, en el directorio WEB-INF/. Se trata de un archivo XML cuyo elemento raíz es web-app.

Como se muestra en la figura A.17 se da un pequeño ejemplo del archivo web.xml:

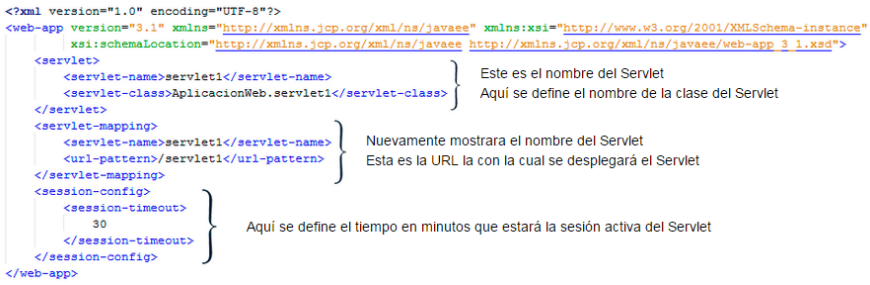

Figura A.17: Descriptor de aplicación web

A.2.2.4. Alcances

- ServletContext: un objeto ServletContext es creado por el contenedor web al momento de desplegar el proyecto. Este objeto puede ser usado para obtener información de configuración del archivo web.xml. Esto es solo un objeto ServletContext por aplicación web.

  Si cualquier información es compartida con muchos Servlets, esto es mejor para proveer del archivo web.xml usando el elemento context-param.

- Session scope: este scope nos permite tener datos que vivirán a lo largo de múltiples peticiones HTTP para un mismo usuario, mientras el usuario esté dentro de la aplicación. Cada usuario verá únicamente sus datos y no habrá forma de que vea los de los demás.

- Request scope: este es el scope más pequeño, los datos asociados con la petición únicamente estarán disponibles mientras se realiza dicha petición.

A.2.2.5. Despachador de peticiones

El Request Dispatcher es una interface, implementación que define un objeto que puede recibir peticiones del cliente y enviárselas a cualquier recurso (tal como HTML, servlets, páginas JSP, imágenes) en el servidor.

El contenedor de servlets crea el objeto RequestDispatcher, que es usado como una envoltura alrededor de un de recurso de servidor localizado en un camino en particular o dando un nombre en particular.

Esta interface está destinada a envolver servlets, pero un contenedor de servlets puede crear objetos RequestDispatcher para envolver cualquier tipo de recurso.

A.2.2.6. Filtros

Los filtros en una aplicación son posibles de definir en serie, porque son los que interceptan las peticiones que son enviadas por parte del usuario. Estos filtros definen patrones de las URLs que desean interceptar. En una aplicación web puede haber múltiples filtros definidos, los cuales serán invocados en el orden por el cual fueron definidos en el descriptor de despliegue de la aplicación.

Los filtros también permiten gestionar el control de acceso, registros (logs), transformar algún contenido, mantener una cache de páginas dinámicas, etc.; también pueden encadenarse creando así una cadena de filtros, pero no todos los filtros están obligados a interceptar todas las peticiones, pero puede ser posible que dos o más filtros hayan indicado que quieren interceptar patrones de URL que en ocasiones son compatibles uno con otro, por lo tanto todos esos filtros deberán ser aplicados a la petición.

A.3. Servlets

Un servlet es una clase del lenguaje de programación java que se utiliza para ampliar las capacidades de los servidores que alojan aplicaciones que son accedidas mediante un modelo de programación de petición y respuesta. Aunque los servlets pueden responder a cualquier tipo de petición, son comúnmente usados para extender aplicaciones alojadas en los servidores web. Para tales aplicaciones, la tecnología java servlets define las clases de servlets HTTP-especificas. Los servlets generan contenido dinámico HTML y corren del lado del servidor orientados a petición-respuesta, esta tecnología pertenece a Java 2 Enterprise Edition (J2EE). Son una herramienta muy importante en la parte de aplicaciones web, ya que con ellos se pueden realizar bastantes tareas, tales como:

- Leer los datos que son enviados por un usuario, usualmente en los formularios web.
- Genera resultados, tales como requerir consultas de bases de datos o invocar a otras aplicaciones.
- Establecer parámetros de respuesta HTTP, para así ordenarle al navegador el tipo de documentos que se le devolverá al usuario. Estos programas son eficientes ya que en ellos se pueden tener N número de peticiones simultáneamente por datos que son enviados por usuarios, tienen una infraestructura muy amplia para tratar automáticamente datos de los formularios HTML los cuales están frecuentemente cambiando y se pueden comunicar con el navegador web para así poder compartir datos y poder mantener datos entre peticiones, ya que las páginas web que usan bases de datos deben de estar actualizadas, para estarnos mostrando constantemente los cambios en los datos o registros. La interface con los usuarios es a través de páginas HTML, es decir una página HTML proporcionara los datos al servlet, lo activa, el servlet recibe los datos, los procesa y le responde al usuario con otra página HTML. Los servlets tienen cuatro propiedades, los cuales son explicados a continuación:
- Manejo de sesiones: se puede hacer seguimiento de usuarios a través de distintos servlets a través de la creación de sesiones.
- Utilización de cookies: las cookies son pequeños datos en texto plano que pueden ser guardados en el cliente. La API de servlets permite un manejo fácil y limpio de ellas.
- Multi-thread: los servlets soportan el acceso concurrente de los clientes, aunque hay que tener especial cuidado con las variables compartidas a menos que se utilice la interfaz SingleThreadModel.
- Programación en java: se obtienen las características de multiplataforma o acceso a APIs como JDBC, RMI, etc.

A.3.1. Arquitectura de un Servlet

La arquitectura de los servlets se define en el paquete javax.servlet en donde se proporciona clases e interfaces para poder escribirlos.

La abstracción central en el API servlet es el interface servlet. Todos los servlets implementan este interface, bien directamente o más comúnmente, extendiendo una clase que lo implemente como HttpServlet.

El interface servlet declara, pero no implementa, métodos que manejan el servlet y su comunicación con los clientes. Los escritores de servlets proporcionan algunos de esos métodos cuando desarrollan un servlet.

Cuando un servlet acepta una llamada de un cliente, recibe dos objetos:

- Un ServletRequest, que encapsula la comunicación desde el cliente al servidor.
- Un ServletResponse, que encapsula la comunicación de vuelta desde el servlet hacia el cliente.

A.3.2. Peticiones HTTP

Las peticiones HTTP es la forma de comunicación entre el cliente (en este caso se refiere a un navegador web) y el servidor que enviara información. La primera parte de la petición es el método que se emplea para realizar la petición (habitualmente GET o POST).

A.3.2.1. La clase ServletRequest

El ServletRequest define un objeto que proporciona información de un servlet a una petición del cliente. El contenedor de servlets crea un objeto ServletRequest y lo pasa como un argumento al método de servlets servicio.

Un objeto ServletRequest proporciona datos incluyendo nombres de parámetros y valores, atributos y un flujo de entrada. Las interfaces que extienden de ServletRequest pueden proporcionar datos específicos de un protocolo.

A.3.2.2. La clase HttpServletRequest

La interface es una extensión de ServletRequest que proporciona información de peticiones para los servlets. El contenedor de servlets crea un objeto HttpServletRequest y lo pasa como un argumento a los métodos de servicio del servlet (doGet, doPost, etc.)

El mensaje de petición es encapsulado en un objeto HttpServletRequest, que es pasado dentro del método doGet ().

La interface HttpServletRequest proporciona muchos métodos para que se recuperen las cabeceras:

- Métodos Generales:
  - getHeader(java.lang.String name): devuelve el valor de la cabecera como una cadena, si la petición no incluye una cabecera con el nombre especificado, el metodo regresa nulo.

- getHeaders(java.lang.String name): regresa una enumeración de todas las peticiones que esta contiene, si esta petición no tiene cabecera, regresa una enumeración vacia.
- getHeaderNames( ): regresa el valor especificado de la cabecera como un int, si la solicitud no tiene cabecera con el nombre especifico el metodo regresa -1.

- URL relacionadas:

  - getRequestURI( ): devuelve parte de las peticiones del protocolo a la cadena de consulta en la primer linea de la petición HTTP
  - getMethod( ): retorna el nombre del metodo HTTP con el que fue hecha la peticion, ya sea GET, POST o PUT.

A.3.3. Respuestas HTTP

Las respuestas HTTP son líneas enviadas por el servidor hacía el navegador. En las respuestas los datos son enviados por el cuerpo de la petición, estas respuestas pueden contener una gran cantidad de datos, los cuales pueden ser enviados.

En las respuestas el mensaje de solicitud está seguro ya que los datos no son expuestos a una URL, este mensaje de solicitud no puede ser marcado, pero si es menos eficiente.

A.3.3.1. La clase ServletResponse

El ServletResponse define un objeto para ayudar un servlet cuando está enviando una respuesta al cliente. El contendor del servlet crea un objeto ServletResponse y lo pasa como argumento al método del servicio servlet.

Esta interface permite recuperar un flujo de salida para enviar datos al cliente, indicar el tipo de contenido que es devuelto por la respuesta; códigos de estado y cookies.

A.3.3.2. La clase HttpServletResponse

HttpServletResponse es una extensión de ServletResponse, el mensaje de respuesta es encapsulado en el HttpServletResponse que es pasado dentro de doGet() por la referencia que es recibida por la salida del servlet, esta extensión permite a los servlets poner el estado y cabeceras.

La respuesta puede incluir tres componentes: un código de estado, cualquier número de cabeceras HTTP y un cuerpo de respuesta.

HttpServletResponse también incluye un número de métodos para el manejo de las respuestas HTTP, a continuación se muestran algunas de ellas:

- setStatus(int sc): este metodo es usado para establecer el código de regreso de la petición.
- sendError(int sc, java.lang.String msg): envía una respuesta al cliente usando el código de estatus y limpiando el buffer.
- sendRedirect(java.lang.String location): envía una respuesta temporal redireccionada al cliente usando la dirección URL especificada.

A.3.4. Ciclo de vida de un Servlet

El ciclo de vida de un servlet se define por tres métodos; init(),service() y destroy(). Cada uno de estos métodos desarrolla una acción diferente para que el servlet desarrolle su función correctamente.

El método init() es diseñado para ser llamado solo una vez, es llamado cuando el servlet es creado y no se le vuelve a llamar de nuevo por cada petición que los usuarios hagan. El método service() es invocado cuando la primera petición llega por parte del cliente, después del método init() para permitir al servlet responder la petición. Este método es el principal y el encargado de realizar las tareas actuales.

El objetivo del método destroy() es eliminar al servlet invocado cuando todas las secuencias dentro del método service() son completadas, y es únicamente llamado cuando llega al final el ciclo de vida del servlet (ver figura A.18).

A.3.4.1. ServletConfig

ServletConfig es implementado por el contenedor de servlets para inicializar un servlet sencillo usando init(). Esto es, pasar los parámetros de inicialización al servlet usando el descriptor de despliegue web.xml.

A.3.4.2. GenericServlet

El GenericServlet implementa las interfaces servlet y ServletConfig y contiene todo el código base requerido por un servlet para mantener su ciclo de vida. Para escribir un GenericServlet es suficiente con sobrescribir el método abstracto service, con esta clase es fácil recibir un modelo y escribir servlets fácilmente.

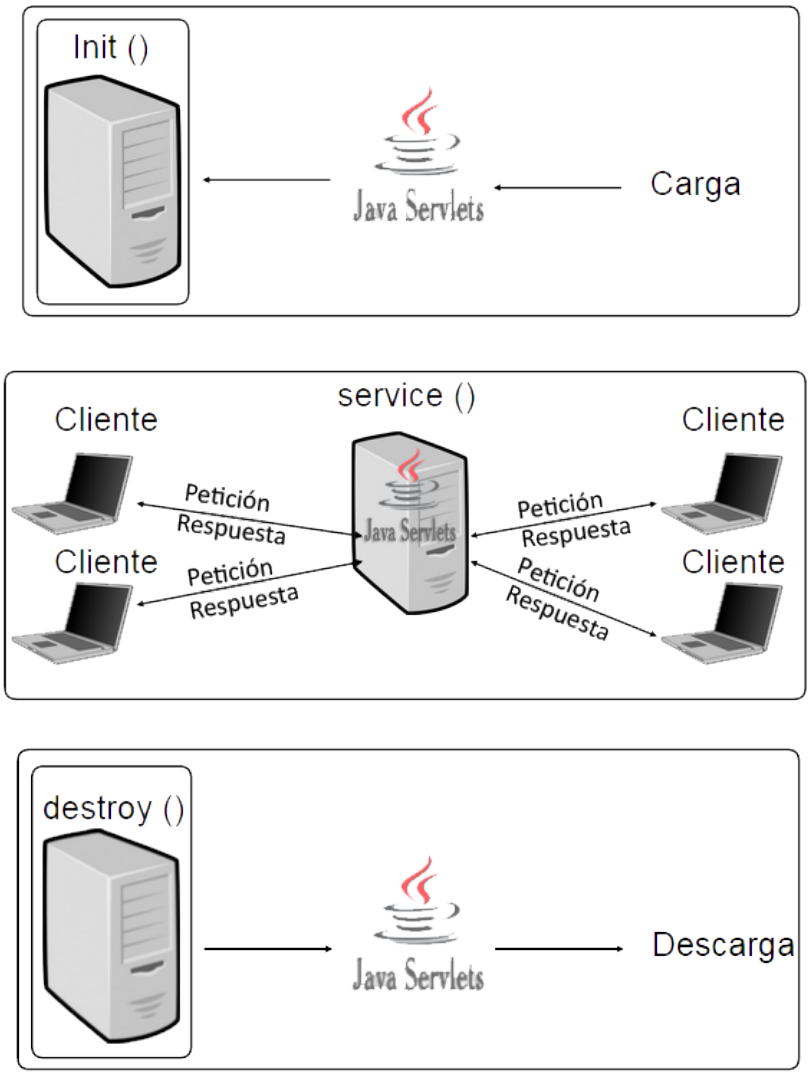

Figura A.18: Ciclo de vida de un servlet

A.3.4.3. HttpServlet

HttpServlet se extiende de la clase GenericServlet para permitir en sí no solo las capacidades básicas de un servlet, sino también el protocolo HTTP especifica otras capacidades como cookies, sesiones, etc. En esta clase se define un protocolo HTTP específico para el servlet.

Referencias

[1] Tim Berners-Lee, Mark Fischetti, and Michael L Foreword By-Dertouzos. *Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor*. HarperInformation, 2000.

[2] W3C HTML Working Group et al. Html 5.0, 2010.

[3] James Goodwill. *Pure JSP Java Server Pages*. Sams, 2000.

[4] Fernando Pech-May, Mario A Gomez-Rodriguez, Salvador U Lara-Jeronimo, and Luis A de la Cruz-Diaz. Hacia el desarrollo de aplicaciones web con tecnologías jpa, ejb, jsf y primefaces. *Instituto Tecnológico Superior de los Ríos*, 2012.

[5] Fernando Berzal, Juan Carlos Cubero, and Francisco J Cortijo. *Desarrollo Profesional de Aplicaciones Web con Asp. net*. iKor Consulting, 2004.

[6] Hypertext preprocessor. `http://php.net/`. recuperado el 10 de abril de 2017.

[7] Robin P Horton, Tamsin Buck, Patrick E Waterson, and Chris W Clegg. Explaining intranet use with the technology acceptance model. *Journal of information technology*, 16(4):237–249, 2001.

[8] Manuel Castells. Internet y la sociedad red. *La factoría*, 14:15, 2001.

[9] Hypertext transfer protocol. `http://www.w3.org/Protocols/`. Recuperado el 17 de Mayo de 2017.

[10] Guía breve de accesibilidad web. `http://www.w3c.es/Divulgacion/GuiasBreves/Accesibilidad`. Recuperado el 18 de Mayo de 2017.

[11] David Flanagan. *JavaScript: the definitive guide*. .º'Reilly Media, Inc.", 2006.

[12] Adrian Kingsley-Hughes, Kathie Kingsley-Hughes, and Daniel Read. *VBScript programmer's reference*. John Wiley & Sons, 2004.

[13] Common gateway interface. https://desarrolloweb.com/articulos/758.php. recuperado el 10 de abril de 2017.

[14] Sistemas distribuidos. http://augcyl.org/?page_id=231. Recuperado el 5 de Abril de 2017.

[15] Apache tomcat 7.0. http://tomcat.apache.org/. Recuperado el 25 de Abril de 2017.

[16] Ramos C. y Siller M. Martínez G., Orozco A. A peer-to-peer architecture for real-time distributed visualization of 3d collaborative virtual environments. Master's thesis, Centro de Investigación y Estudios Avanzados del IPN, Guadalajara, México, 2009.

[17] Transmission control protocol. https://www.ibm.com/support/knowledgecenter/es/ssw_aix_72/com.ibm.aix.networkcomm/tcpip_protocols.htm. recuperado el 10 de abril de 2017.

[18] J2ee. http://www.oracle.com/technetwork/java/javaee/overview/index.html. Recuperado el 25 de Abril de 2017.

[19] Alex Berson. *Client-Server Architecture*. 2nd. edition,McGraw-Hill, Universidad de Michigan, USA, 2007.

[20] George F. Coulouris and Jean Dollimore. *Distributed systems: concepts and design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1994.

[21] Llamadas a procedimientos remotos. http://www.arcos.inf.uc3m.es/~infosd/lib/exe/fetch.php?media=es:t6_llamadas_a_procedimientos_remotos.pdf. Recuperado el 22 de Marzo de 2017.

[22] Andrea Mesa Múnera. Método para el manejo del balanceo de carga en ssistemas de cómputo distribuido de alto desempeño. *Vol. 1*, pages 1–50, Octubre 2009.

[23] H. Castillo Zacatelco. Balance de carga en sistemas distribuidos. *Vol. 7*, pages 1–10, Octubre 2007.

[24] Load balancing. http://tutorials.jenkov.com/software-architecture/load-balancing.html. Recuperado el 13 de Marzo de 2017.

[25] Distribución de carga. http://www.cuaed.unam.mx/boletin/boletinesanteriores/boletinsuayed01/balanceo.php. Recuperado el 25 de Marzo de 2017.

[26] Abraham. Silberschatz. *Fundamentos de Bases de Datos.* Quinta Edición, Mc Graw-Hill, España, 2006.

[27] T. Kindberg G. Coulouris, J. Dollimore and G. Blair. *Distributed Systems: Concepts and Design.* Quinta Edición, Addison-Wesley, USA, 2011.

[28] Carlos Coronel. *Bases de datos.* Segunda Edición, México, 2004.

[29] Rubyrep. `http://www.rubyrep.org/`. Recuperado el 15 de Enero de 2017.

[30] Slony-i. `http://slony.info/`. Recuperado el 15 de Enero de 2017.

[31] Symmetricsds. `http://www.symmetricds.org/`. Recuperado el 15 de Enero de 2017.

[32] Pgpool-ll. `http://www.pgpool.net/mediawiki/index.php/Main_Page`. Recuperado el 15 de Enero de 2017.

[33] Bucardo. `https://bucardo.org/`. Recuperado el 15 de Enero de 2017.

[34] Daffodil replicator. `https://code.google.com/p/replicatoros/`. Recuperado el 15 de Enero de 2017.

[35] Shamkant B. Navathe. ELMASRI, Navathe. *Fundamentos de Sistemas de Bases de Datos.* traducción, Verónica Canivell Castillo.

[36] Tamer. ÖZSU. *Principles of distributed database system.* Segunda edición, Prentice-Hall International Edition, USA.

[37] Elmasri RAMEZ. *Fundamentos Sistemas de Bases de Datos.* Tercera Edición, Pearson educación, Madrid, 2006.

[38] J2sdk. `http://www.oracle.com/technetwork/java/archive-139210.html`. Recuperado el 25 de Abril de 2017.

[39] Mark H Goadrich and Michael P Rogers. Smart smartphone development: ios versus android. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 607–612. ACM, 2011.

[40] Grady Booch, Bruce Eckel, and Martin Fowler. Análisis y diseño orientado a objetos con aplicaciones (2da edición). 1996.

[41] Inderjeet Singh and Mark Johnson. *Designing enterprise applications with the J2EE platform.* Addison-Wesley Professional, 2002.