UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

Análisis de desempeño de una arquitectura de software para aplicaciones Web.

T E S I S

QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

P R E S E N T A:
H E R I B E R T O G A L D A M E Z T O R I J A

Director de Tesis:
Dr. Jorge Luis Ortega Arjona
Facultad de Ciencias, UNAM

Ciudad Universitaria, CD. MX. Agosto, 2016

Agradecimientos

A mis padres, Araceli Torija Rentería y Oudulio Galdamez Toribio, por creer en mí sin condiciones y darme las oportunidades para seguir creciendo como persona. Por haberme guiado y apoyado en todo momento. Por su cariño y comprensión. Tuve la gran fortuna de tenerlos como padres, este trabajo es para ustedes, gracias.

A toda mi familia, mis hermanos Ricardo y Julio Adrián, que siempre me han brindado su apoyo. A mis amigos de la infancia, por su confianza y lealtad que me otorgan, su valiosa amistad es un pilar para lograr mis objetivos, a Víctor Colín, Ernesto Rosete , Alejandra y Marco López, a todos ellos, muchas gracias.

A mis compañeros en el posgrado, por brindarme su amistad antes y durante esta Maestría, por el apoyo académico y mi formación como persona, por haber compartido momentos muy singulares y particulares, a Fabiola, Karen, Mario, Octavio, Alex, Aurelio, Victor, Aitor, Eduardo, Ernesto, en fin, a todos los compañeros que son un aliento para el estudio.

Agradezco con mucho respeto y admiración al Dr. Jorge Luis Ortega Arjona, por haberme dado la oportunidad de trabajar bajo su tutela, su guía y comprensión durante mi estancia en el posgrado, sus consejos y enseñanzas, y por ser una persona muy comprometida con su trabajo. Gracias.

A la Universidad Nacional Autónoma de México, por la formación que he recibido durante el bachillerato, licenciatura y maestría, por permitirme estudiar y lograr mis objetivos como persona. Agradeciendo de igual manera a CONACYT, por proporcionarme un invaluable apoyo económico para poder exigirme al máximo.

Índice general

Índice de Figuras

Índice de Tablas

Resumen

Hoy en día, las aplicaciones Web se han convertido en un parte fundamental de la vida diaria; se realiza un uso intensivo de ellas especialmente desde los dispositivos móviles y su buen desempeño es esencial. Cada petición que se le realiza a una aplicación Web puede transformarse en una tarea atendida por la aplicación. Una petición solicitada a una aplicación Web que tarda un tiempo considerable en ser atendida puede afectar su desempeño. Diseñar aplicaciones Web que puedan atender una petición en un tiempo aceptable requiere un análisis detallado.

La presente tesis propone un diseño de una Arquitectura de Software de una aplicación Web, que permita a la aplicación mantener la calidad en el servicio a través de tiempos de respuesta aceptables a las peticiones realizadas. Se basa en patrones y modelos que ya existen, pero que no fueron diseñadas para comunicarse entre ellas, o son variaciones para manejar el desempeño en una aplicación Web. Se presenta una implementación y evaluación del diseño propuesto en un lenguaje de programación orientado a objetos.

Capítulo 1

Introducción

1.1. El contexto de la tesis

La Web es el conjunto de toda la información accesible a través de computadoras conectadas mediante una o varias redes [6]. La constante evolución de las tecnologías relacionadas con la Web permite compartir y utilizar una aplicación completa por Internet. Una aplicación Web es un conjunto de programas que permite a diferentes clientes enviar y recibir información a través de Internet [30]. Está formada por aplicaciones con diferentes componentes y subsistemas que se ejecutan en entornos separados y en diferentes plataformas conectadas por red.

Por otro lado, una Arquitectura de Software es una descripción de los subsistemas y componentes de un Sistema de Software, y las relaciones que existen entre ellos [7]. La Arquitectura de Software es la primera etapa en la creación de Software donde se pueden tratar los atributos de calidad de un Sistema de Software [4].

El desempeño es un atributo de calidad que mide la velocidad y eficiencia en que un Sistema de Software puede completar ciertas tareas de computación [22]. Por lo cual, es un atributo deseable en cualquier aplicación Web. El desempeño de una aplicación Web se deteriora rápidamente cuando la carga de trabajo aumenta [22].

1.2. Problemática

Una petición a una aplicación Web consume recursos computacionales. Los recursos pueden llegar a agotarse y provocar fallas en el servicio. Una falla en el servicio es un resultado incorrecto con respecto a su especificación o un comportamiento inesperado percibido por los usuarios del Sistema [15]. Por ejemplo, si el sistema no responde en un tiempo acotado se percibe como un error por parte del usuario.

Uno de los retos en el desarrollo este tipo de aplicaciones recae en el diseño de componentes que manejen el tiempo de respuesta. El tiempo de respuesta es una métrica de rendimiento que se utiliza en un sistema de Software para medir la velocidad de respuesta a las peticiones de los usuarios interactivos [22].

Un reto adicional en el desarrollo de aplicaciones Web de alta concurrencia es que puede recibir peticiones a cualquier hora y de cualquier lugar. Por lo cual no es posible predecir un comportamiento de la aplicación. Esto es provocado por el ambiente no determinista de la red, que es una limitación que establece que la aparición de hechos, eventos, o la secuencia en la que aparecen en un sistema de Software no son totalmente determinables [32].

1.3. Hipótesis

La presente tesis pretende dar respuesta a la siguiente pregunta:

"¿Es posible proponer una Arquitectura de Software para una aplicación Web, mediante patrones, que permita dividir el trabajo que se realiza en la atención de peticiones, en tareas más simples que se pueden ejecutar de manera simultánea, con el objetivo de mejorar su desempeño?"

1.4. Aproximación

El enfoque del presente trabajo de tesis es definir una Arquitectura de Software para atender el problema de desempeño en aplicaciones Web, proporcionando un diseño que permita dividir el trabajo a realizar en tareas más simples que se pueden ejecutar de manera concurrente.

La concurrencia es un término que se refiere a la familia de políticas y mecanismos que permite a uno o más hilos o procesos ejecutar sus tareas de procesamiento simultáneamente[5]. El rendimiento de una Arquitectura de Software puede ser mejorada mediante el aprovechamiento de la capacidades de procesamiento paralelo [22].

La forma de poner a prueba la efectividad de la solución es mediante la implementación de los patrones propuestos en algún lenguaje de programación. Mediante una simulación de peticiones a la Arquitectura se generan los escenarios para los cuales se ofrece una solución concurrente.

1.5. Contribuciones

El objetivo principal de este trabajo de tesis es describir una Arquitectura de Software, mediante patrones, para una aplicación Web que realice procesamiento en paralelo para atender las peticiones que sean realizadas. La arquitectura representa una alternativa a problemas de rendimiento que promueve este tipo de desarrollo. Considerando lo anterior, las contribuciones principales son las siguientes:

1. Se describe una Arquitectura de Software para una aplicación Web que permite la división del trabajo en tareas más simples que se pueden ejecutar de manera simultánea en la atención de peticiones.

2. Se propone utilizar un modelo de concurrencia computacional existente como unidad de procesamiento principal.

3. Se demuestra que la Arquitectura de Software propuesta mejora el desempeño de una aplicación Web.

1.6. Estructura de la tesis

La estructura de la tesis es la siguiente:

- **Capítulo 2.** En este capítulo se introducen los conceptos relacionados con esta tesis:

Una introducción al concepto de Arquitectura de Software, los objetivos que trata de resolver y restricciones. También una introducción a los Patrones de Software, la manera en cómo son descritos y su relación. A continuación se describen los atributos de calidad del Software, las relaciones que existen entre ellos y la importancia que tienen en el Diseño de Software. Finalmente se describe el modelo de programación Map-Reduce, que permite simplificar la lógica de paralelización de tareas dentro de una aplicación

- **Capítulo 3.** En este capítulo se presenta un resumen del trabajo relacionado a los temas de interés de la presente tesis. Se presentan trabajos acerca del diseño, desarrollo y evaluación de aplicaciones Web relacionados con el desempeño donde se resaltan las características más importantes.

- **Capítulo 4.** Se describe la Arquitectura de Software propuesta mediante patrones. Primero se plantea una Arquitectura de Software para una aplicación Web con el patrón arquitectónico Layers [7]. Posteriormente se describe la arquitectura propuesta para mejorar el desempeño de una aplicación Web con el patrón arquitectónico Parallel Layers [24].

- **Capítulo 5.** En este capítulo se propone un problema a resolver para ambas arquitecturas descritas en el Capítulo 4. Se describen las características del algoritmo propuesto como solución. Se realiza una implementación y pruebas de las dos arquitecturas con el objetivo de compararlas. Finalmente se establecen las características de los escenarios de pruebas para ambas arquitecturas.

- **Capítulo 6.** Se analizan los resultados de las pruebas realizadas a las aplicaciones Web. Se proponen métricas y un método de medición para poder evaluar el desempeño de una aplicación Web y se realiza la comparación de los resultados obtenidos de las implementaciones que se realizan en el Capítulo 5.

- **Capítulo 7.** En este capítulo se presenta una discusión y análisis de los resultados obtenidos, así como los posibles trabajos a realizar en un futuro.

Capítulo 2

Antecedentes

En este capítulo se mencionan algunas características de las aplicaciones Web y su relación con la arquitectura Cliente - Servidor. Posteriormente se mencionan patrones y enfoques arquitectónicos para el diseño de Software. A continuación, se describen algunos atributos de calidad de Software y sus características. Finalmente, se describe el modelo de programación Map-Reduce y su funcionamiento.

2.1. Aplicaciones Web

Las aplicaciones Web se pueden considerar como un conjunto de archivos de hipertexto vinculados que presentan información con uso de texto y gráficas limitadas. Debido a su constante crecimiento y el progreso en las arquitecturas basadas en Internet, también están integradas con bases de datos corporativas y aplicaciones de negocios [26]. Una aplicación Web es aquélla que los usuarios usan accediendo a un servidor Web a través de Internet, mediante una arquitectura Cliente - Servidor.

2.1.1. Cliente - Servidor en aplicaciones Web

Un cliente Web es un programa que interactúa con el usuario para solicitar recursos a un servidor. Un navegador de Internet es el ejemplo más representativo de un cliente Web. Un servidor Web es un programa que espera de forma permanente peticiones de los clientes Web. Los clientes requieren de recursos que son administrados por el servidor Web. Algunos ejemplos de los recursos son hardware, software o

información. En la figura 2.1 se muestra un diagrama a bloques de una arquitectura Cliente-Servidor en una aplicación Web. Un ejemplo de un proceso para realizar una petición de un cliente hacia un servidor Web es el siguiente:

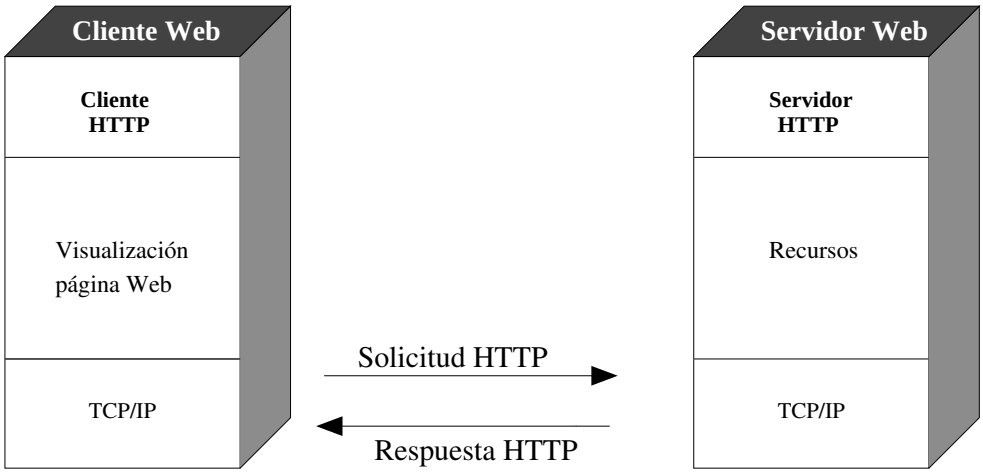

Figura 2.1: Diagrama de bloques de una arquitectura Cliente-Servidor

1. El usuario especifica en el cliente Web la dirección URL del recurso que desea obtener. Una URL es una cadena que identifica a un recurso en la red que puede cambiar en el tiempo [6].

2. El cliente Web realiza una petición al servidor mediante el protocolo HTTP. El protocolo HTTP define la estructura de los mensajes y como se intercambian entre un cliente y un servidor Web [20]. HTTP forma parte de la familia de protocolos de comunicación TCP/IP [15].

3. El cliente Web establece una conexión con el servidor Web.

4. El cliente Web solicita el recurso deseado. Un ejemplo de un recurso es una página Web. Una página Web es un archivo HTML, un lenguaje de marcas que se utiliza para representar el contenido publicado en la Web [6].

5. El servidor Web responde mediante el protocolo HTTP con el recurso solicitado, o un código de error en caso de que el recurso no exista o no se tenga acceso a él.

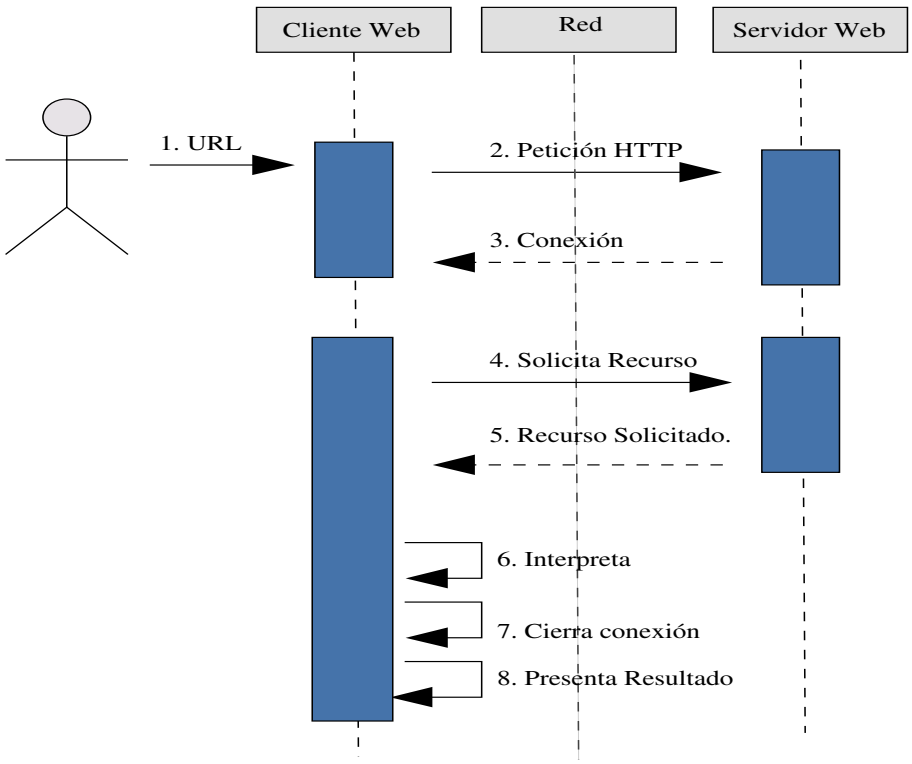

Figura 2.2: Diagrama de secuencia del ejemplo cliente - servidor

6. El cliente Web recibe la respuesta a la petición. Inicia las tareas para la interpretación del recurso obtenido. En este ejemplo el cliente debe de interpretar HTML para presentar el recurso al usuario. Existen otro tipo de recursos que se pueden recibir por parte de un servidor Web. Un ejemplo de ello es XML, un lenguaje de marcas extendido.

7. Se cierra la conexión entre el cliente Web y el servidor Web.

8. Finalmente, el cliente Web muestra el resultado de la petición al usuario.

En la figura 2.2 se muestra el diagrama de secuencia del proceso anterior. El cliente utiliza HTTP como protocolo de comunicación. Existen otros clientes Web que utilizan diferentes protocolos. Ejemplos de ellos son el protocolo File Transfer Protocol (FTP) que se utiliza para realizar transferencia de archivos o el protocolo para la transferencia simple de correo electrónico (SMTP).

2.1.2. Atributos de las aplicaciones Web.

Aunque existe una gran variedad de aplicaciones Web, dependiendo del contexto en que sean implementadas, existen atributos comunes en la mayoría de ellas. Algunos de estos atributos se listan a continuación [26] :

1. **Uso de las redes.** Una aplicación Web reside en una red y debe atender las necesidades de diversos clientes.

2. **Concurrencia.** Pueden acceder un gran número de usuarios a la vez. La forma en que es utilizada varía de acuerdo al tipo de usuario.

3. **Carga impredecible.** El número de usuarios cambia en diferentes órdenes de magnitud de un momento a otro.

4. **Desempeño**. Si un usuario espera demasiado por la respuesta a una petición, el usuario lo percibe como un error o un comportamiento inesperado.

5. **Disponibilidad**. Frecuentemente los usuarios demandan el acceso a una aplicación Web las 24 horas de los 365 días del año.

6. **Orientado a datos**. Existen aplicaciones Web que se utilizan para acceder a información que no es parte integral del ambiente basado en Web. Un ejemplo de ello son las aplicaciones financieras.

7. **Seguridad.** Con el fin de proteger contenido sensible y proporcionar formas seguras de transmisión de los datos, deben implementarse mecanismos de seguridad con el apoyo de la infraestructura de la aplicación Web y dentro de la misma aplicación.

2.1.3. Entorno de las aplicaciones Web.

Las aplicaciones Web se pueden emplear en tres entornos infórmaticos similares: Internet, intranet y extranet [23]. Internet es una red global que conecta a millones de computadoras a nivel mundial con el objetivo principal de distribuir y compartir información [23].

Por otra parte, intranet es una red de computadoras basadas en los protocolos que gobiernan Internet (TCP/IP) que pertenece a una organización y es accesible únicamente por los miembros de la misma organización con el mismo objetivo de compartir y distribuir información [23].

El entorno extranet es parecido al de intranet. La diferencia consiste en que personas autorizadas ajenas pueden acceder parcialmente a la red de la organización de acuerdo con un nombre de usuario y contraseña asignados. Este tipo de redes son utilizadas por empresas que colaboran para compartir información entre ellas [23].

Los avances en el cómputo móvil y las comunicaciones inalámbricas más la amplia adopción a nivel mundial de los dispositivos móviles están permitiendo a un creciente número de usuarios acceder a Internet [29]. Los clientes de dispositivos móviles han generado un entorno del cual podemos mencionar algunos factores que han influido en el desarrollo de aplicaciones Web [29]:

1. **Diversidad de clientes.** Cada usuario puede utilizar un dispositivo diferente para demandar contenido. Esto implica una diversidad de dispositivos. Cada dispositivo tiene su propio formato de pantalla. El soporte de hardware y software es particular del dispositivo y cada uno puede conectarse a una velocidad de acceso distinta.

2. **Número de usuarios.** El número de usuarios que acceden a una aplicación Web es impredecible y varía de forma considerable que puede provocar un problema de desempeño. Un ejemplo es el número de peticiones realizadas a una aplicación Web de comercio electrónico ocasionadas por una promoción de productos o algún evento significativo.

3. **Variedad de contenido.** El cliente Web puede solicitar una amplia variedad de contenido. Algunos ejemplos son gráficas, imágenes, datos, audio y video. En ocasiones existe contenido que se debe adecuar a algún estándar de algún país, el idioma o un sistema de unidades son ejemplo de ello.

4. **Diferentes tipos de aplicaciones Web.** Los puntos anteriores pueden provocar que se generen diferentes tipos de aplicaciones dirigidas a ofrecer un servicio en específico. Desde proporcionar paginas Web estáticas o dinámicas,

herramientas colaborativas como redes sociales, la realización de transacciones financieras o un servicio de datos mediante una API (interfaz de programación de aplicaciones) que proporciona datos para la creación de nuevas aplicaciones basadas en los datos obtenidos.

2.1.4. Servicios Web

Un servicio Web es un sistema de software que permite la interacción entre un cliente y un servidor que pueden comunicarse a través de Internet [10]. Los servicios Web son independientes del lenguaje y plataforma. Los clientes se desarrollan utilizando diferentes lenguajes, se pueden ejecutar en diferentes plataformas y comunicarse con un mismo servicio Web. Los clientes de los servicios Web pueden ser otros servicios Web o aplicaciones interactivas como navegadores Web o aplicaciones móviles que utilizan la información proporcionada para ejecutar otro tipo de tareas.

Los servicios Web se pueden clasificar con base en el estilo arquitectónico utilizado para su desarrollo. Una clasificación para los servicios Web basado en su estilo arquitectónico es [28]: los servicios Web basados en SOAP y los servicios Web basados en REST. Una definición de SOAP es [10] :

> "SOAP (Service Oriented Architecture Protocol) proporciona un marco de trabajo estándar, extensible y organizado para la envoltura e intercambio de mensajes XML."

Los métodos de un servicio Web Basado en SOAP se exponen mediante una interfaz. La descripción de las funciones y del intercambio de mensajes entre un cliente y un servidor se basa en el lenguaje WSDL (Web Service Description Language). El cliente debe conocer la ubicación y la descripción del servicio para poder invocar un método del servicio Web [28].

Los servicios Web REST se basan en los principios de la arquitectura REST (Representation State Transfer). Un Servicio Web REST expone su funcionalidad como un conjunto de recursos. Un recurso es cualquier pieza de información que puede ser el objetivo de una interacción y se define como un objetivo conceptual de una referencia de hipertexto [13].

2.1.5. Arquitectura REST

La Arquitectura REST es una arquitectura para sistemas hipermedia distribuidos. Hipermedia es una forma de contenido, por la cual se realiza un proceso de comunicación, se solicita bajo petición y opera textualmente [33]. Los recursos están expuestos por servidores y consumidos por clientes que utilizan métodos HTTP. Se accede a un recurso a través de una dirección URL y su estado se transfiere utilizando su representación [13]. Una característica clave de una interfaz REST es la división del estado de la aplicación entre el cliente y el servidor . A continuación se mencionan algunas características esenciales de REST [21]:

- **Recursos y URL.** En una interfaz REST, todo es un recurso. Todo lo que se puede representar por una URL puede ser un recurso. Un recurso puede ser estático o puede cambiar con el tiempo. El recurso también puede representar una lista de recursos. Aunque todos los recursos deben tener una URL, REST no impone ningún esquema para la construcción de URLs. Es preferible direcciones URL legibles para el humano. Los autores recomiendan que la dirección URL no debe contener ninguna información sobre cualquier operación aplicable al recurso.

- **Conectividad.** Un cliente necesita conocer las direcciones de todos los recursos necesarios para consumir un servicio REST. Las direcciones pueden considerarse identificadores. El servidor debe guiar al cliente sobre los recursos significativos y cercanos al actual para que el cliente pueda tomar la decisión de la ruta a elegir. El servidor puede enviar enlaces a otros recursos hipermedia relacionados.

- **Métodos REST.** Los métodos HTTP más utilizados en los servicios REST son get, post, put y delete. Definen las operaciones de lectura, creación, actualización y eliminación que se pueden aplicar a los recursos. El método get puede considerarse seguro. Implica la consulta de información y recuperación de datos sin efectos secundarios. Con los métodos get, put y delete no importa si la operación se aplica una o varias veces en un recurso, el resultado final siempre es el mismo.

- **Sin Estado.** Cualquier estado de la aplicación se almacena sólo en el cliente. El servidor sólo almacena los recursos y sus estados. Cada petición de un recurso debe contener toda la información necesaria para llevarla a cabo. El servidor no necesita saber de cualquier petición anterior o posterior realizada por el cliente. Cada petición debe suceder en completo aislamiento de cualquier otra petición.

- **Interfaz Uniforme.** Las interfaces de los servicios REST exponen los recursos con un conjunto de métodos HTTP fijo y devuelven valores para cada recurso. Todo es diseñado en términos de recursos en vez de funcionalidad.

En la tabla 2.1 se muestra una comparación de algunas características entre los servicios Web REST y SOAP:

Tabla 2.1: Comparación características servicios Web REST y SOAP

Concepto	***REST***	***SOAP***
Tipo Operación	El protocolo indica el tipo de operación sobre el recurso	El contenido del mensaje decide el tipo de operación
Estándares	No existe actualmente un estándar	Estandares bien definidos, por ejemplo, WSDL
Restricciones en la cantidad de datos que se transmiten en una comunicación	No existen restricciones	Debe de cumplir con el esquema SOAP
Seguridad	La seguridad se puede establecer mediante el protocolo (por ejemplo, HTTPS, SSL)	Un amplio número de estandares establecidos para seguridad
Manejo de errores	Manejo del estatus de la respuesta mediante el protocolo HTTP	No cuenta con un manejo de errores establecido

2.2. Arquitectura de Software

2.2.1. Definición

Existe una amplia variedad de definiciones de Arquitectura de Software. Cada autor se enfoca en resaltar los elementos que le interesan. A continuación se presenta una lista de definiciones de Arquitectura de Software que son la base para obtener una definición que se utiliza en el presente trabajo de tesis.

1. "Es una descripción de los subsistemas y componentes de un sistema de software y las relaciones entre ellos. Una relación indica una conexión entre los componentes. Una relación puede ser estática o dinámica. Una relación estática se refiere a la colocación de los componentes dentro de una arquitectura. Una relación dinámica tratan de conexiones temporales y la interacción dinámica entre los componentes"[7].

2. "La arquitectura de software de un programa o de un sistema de computación es la estructura o estructuras del sistema, que comprende elementos de software, las propiedades visibles de manera externa de esos elementos, y las relaciones entre ellos"[4].

3. Los autores Perry y Wolf presentan el siguiente modelo [25] :

   "Arquitectura de software = [Elementos, Forma, Razón]"

   Donde:
   "Elementos: La arquitectura de software consiste en 3 tipos de elementos: elementos de procesamiento, elementos de datos y elementos de conexión.
   Forma: La forma de una arquitectura de software son las limitaciones impuestas a la implementación. Esto puede relacionarse con las propiedades de elementos particulares o las relaciones entre ellos.
   Razón: La razón o razones que se utilizan para elegir entre diferentes elementos y formas."

4. "Conceptos fundamentales o propiedades de un sistema en un entorno que se establece por sus elementos, las relaciones entre ellos y los principios de su diseño y evolución"[11].

De manera general, las definiciones anteriores se refieren a que la Arquitectura de Software es un enfoque en el razonamiento sobre los decisiones estructurales de un sistema para satisfacer las restricciones y requerimientos establecidos.

La definición que se utiliza en esta tesis es: La Arquitectura de Software es una descripción de la forma de distribución y orden de los elementos de software de un sistema y las relaciones entre ellos. Las relaciones pueden describir la ubicación o comportamiento de los elementos dentro de una arquitectura.

Una de las principales razones para el estudio y el uso de la Arquitectura de Software es cumplir con ciertos objetivos. Estos pueden estar vinculados directamente a los requerimientos definidos por un cliente o pueden ser atributos deseados definidos por una organización en desarrollo. Los atributos son algunas características que el software debe tener en cierta medida. Estos atributos también son conocidos como atributos de calidad del Software [3].

2.2.2. Patrón de software

Un patrón es la descripción de una solución a un problema que ha sido implementada con éxito. La solución es documentada con el objetivo de compartir el conocimiento adquirido mediante experiencias anteriores y que sirva como una guía para resolver problemas similares [7]. El patrón ofrece una descripción generalizada de una solución. La solución se puede aplicar independientemente de alguna tecnología en particular. El arquitecto Christopher Alexander define el término patrón de la siguiente manera:

"Un patrón es una regla de tres partes que expresan una relación entre un contexto, la descripción de la situación donde ocurre el problema, un problema, la especificación de lo que se desea resolver; y una solución, que es el cómo se resuelve el problema"[2].

La definición anterior es utilizada como base para especificar el término de patrón de Software. Sin embargo, existen varias definiciones de patrones de software. A continuación se presentan dos de ellas.

1. "Un patrón para una arquitectura de software describe un problema particular recurrente que surge en un contexto específico de diseño. Proporciona una

descripción de una solución que ha sido probada. La descripción incluye los elementos de los que se constituye, las responsabilidades y relaciones, y la forma en la que interactúan entre ellos" [7].

2. "Los patrones de diseño son descripciones de comunicación de objetos y clases que se adaptan para resolver un problema general de diseño en un contexto particular" [14].

Existen tres conceptos comunes en las definiciones mencionadas: Un problema, que aparece en un contexto, y una solución. Los patrones de software se pueden considerar como una herramienta en el desarrollo de un sistema. El patrón de software ofrece una solución general a un problema de diseño en un contexto dado, sin ofrecer a detalle el diseño completo de un sistema.

2.2.3. Descripción de un patrón de software

Una forma de describir un patrón de software es a través de una plantilla o formulario. La base de las plantillas para los patrones de software también se obtienen del trabajo de Christopher Alexander [2]. Algunos de los principales objetivos de las plantillas que describen patrones de software son: el describir el problema, describir el contexto del problema, describir la solución y describir las ventajas y desventajas del patrón. Para esta tesis se utiliza la forma POSA [7] :

- *Nombre:* El nombre y un breve resumen del patrón.
- *También conocido como:* Otro nombre con el cual es conocido el patrón en caso de que exista.
- *Ejemplo:* Un ejemplo del mundo real que demuestra la existencia del problema y la necesidad del patrón.
- *Contexto:* Situación o situaciones en las cuales se puede aplicar el patrón.
- *Problema:* El problema que resuelve el patrón incluyendo una discusión de las fuerzas.

- *Solución:* El principio esencial de la solución que establece las bases para el patrón.
- *Estructura:* Una especificación detallada de los aspectos estructurales del patrón.
- *Dinámica:* Escenarios típicos que describen el comportamiento en tiempo de ejecución del patrón. Los escenarios pueden ilustrarse con diagramas de secuencia entre objetos.
- *Implementación:* Guía para la implementación del patrón. La cual presenta solo sugerencias.
- *Ejemplo Resuelto:* Discusión sobre cualquier aspecto importante para resolver el ejemplo que no se haya cubierto con las secciones de solución, estructura, dinámica e implementación.
- *Variantes:* Descripción breve de las especializaciones o variantes del patrón.
- *Usos conocidos:* Ejemplos del uso del patrón obtenidos de sistemas existentes.
- *Consecuencias:* Descripción de los beneficios y desventajas que provee el patrón.
- *Véase también:* Referencias a patrones que resuelven problemas similares, y referencias a patrones que ayudan a refinar el patrón que se está describiendo.

2.2.4. Categorización de los patrones de software

Las clasificaciones más conocidas de los patrones de software son con respecto a su nivel de abstracción [7] o su objetivo [14]. Para este trabajo de tesis, se considera la clasificación de los patrones con respecto a su abstracción. La clasificación es la siguiente:

- *Patrones Arquitectónicos.* Describen un esquema estructural fundamental de la organización del sistema. Proveen un conjunto predefinido de subsistemas, especifican sus responsabilidades, e incluyen reglas y guías para organizar las relaciones entre ellos [7].

- *Patrones de diseño.* Proveen un esquema para refinar los componentes de un sistema, las relaciones y comunicaciones entre ellos [14].
- *Modismo.* Es un patrón de bajo nivel especifico de un lenguaje de programación. Describe como implementar aspectos particulares de los componentes o relaciones entre ellos, utilizando características particulares del lenguaje [7].

2.2.5. Ejemplo de un patrón arquitectónico — Parallel Layers

El patrón Parallel Layers [24] es una extensión del patrón Layers [7] con elementos de paralelismo funcional. En este tipo de paralelismo, dos o mas componentes de la capa pueden existir de manera simultánea. Los componentes normalmente ejecutan la misma tarea. Pueden ser creados de manera estática o dinámica.

Contexto

Para este patrón debemos considerar las siguientes restricciones:

- El problema a resolver expresado en algoritmos y en datos.
- Son conocidos la plataforma paralela y el ambiente de programación a utilizar. Los cuales proporcionan un nivel de paralelismo razonable en términos del número de procesadores o ciclos en paralelo disponibles.
- El lenguaje de programación a utilizar. Basado en algún paradigma específico y con un compilador disponible para la plataforma paralela.

Problema.

Un algoritmo está compuesto por dos o más sub-algoritmos más simples. Los cuales pueden ser divididos a su vez en otros sub-algoritmos de manera recursiva. Creciendo en forma de una estructura de árbol hasta que se alcance alguna de las hojas. En las hojas se encuentra la sub-parte más simple del algoritmo. La comunicación entre las diferentes capas es estrictamente en forma de árbol. Los datos pueden ser divididos cuando se transportan entre capa y capa, sin ser estrictamente

dependientes. Generalmente el desempeño en términos del tiempo de ejecución es la característica de interés.

Solución.

Una forma de solucionar el problema es utilizando el paralelismo funcional para ejecutar los sub-algoritmos. Permitiendo la existencia y ejecución simultánea de más de una instancia de un componente de capa a través del tiempo. Cada una de las instancias pueden ser compuestas por más sub-algoritmos, más simples. La ejecución de una operación en un sistema de capas involucra la ejecución de un conjunto de operaciones en varias capas subsecuentes. Las operaciones son usualmente provocadas por una llamada entre capas. Los datos son compartidos de manera vertical entre las capas en forma de argumentos en cada una de las llamadas. Usualmente, cada capa tiene que esperar el resultado de las capas inferiores. Si cada capa está representada por más de un componente, pueden ser ejecutadas en paralelo y atender nuevas solicitudes. Varios conjuntos de operaciones ordenadas pueden ejecutarse por el mismo sistema de forma simultánea y varios cálculos pueden coincidir en el tiempo.

Estructura.

En este patrón arquitectónico, las diferentes operaciones son ejecutadas por entidades conceptuales que son independientes y ordenadas en forma de capas. Cada capa está compuesta por varios componentes que ejecutan la misma operación. La comunicación entre capas se realiza mediante llamadas. Cada capa se refiere a otra como una combinación de componentes. La misma operación se ejecuta por diferentes grupos de componentes relacionados funcionalmente. Los componentes existen y operan de manera simultánea en tiempo de ejecución. En la figura 2.3 se muestra un diagrama de objetos de la estructura del patrón Parallel Layers [24].

Dinámica.

Todos los componentes están activos al mismo tiempo. Cada uno aceptando y procesando peticiones. Después de cada proceso de petición, los componentes pueden

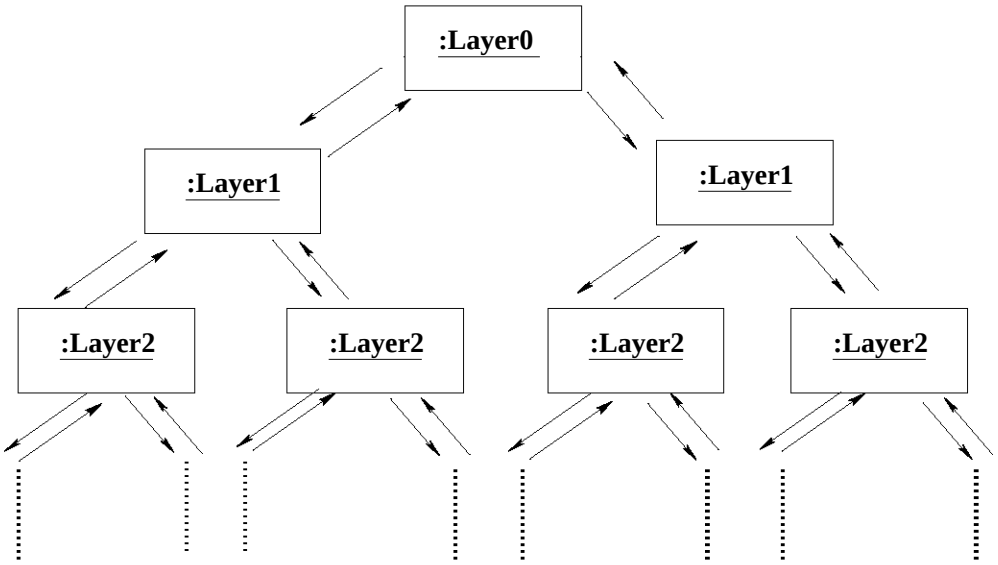

Figura 2.3: Diagrama de objetos de la estructura del patrón Parallel Layers

regresar un resultado o realizar otra llamada a otro componente de capa inferior. Si una nueva petición ocurre en la capa superior inicial (Layer0) un componente libre de la primer capa la atiende.

Se presenta un escenario para ejemplificar el comportamiento de dos cálculos a realizar. El cálculo 1 requiere realizar la operación Op.A, la cual requiere de la evaluación de Op.B, que a su vez requiere de la evaluación de Op.C. El cálculo 2 necesita solo las operaciones Op. A y Op. B. La ejecución en paralelo se muestra en la figura 2.4.

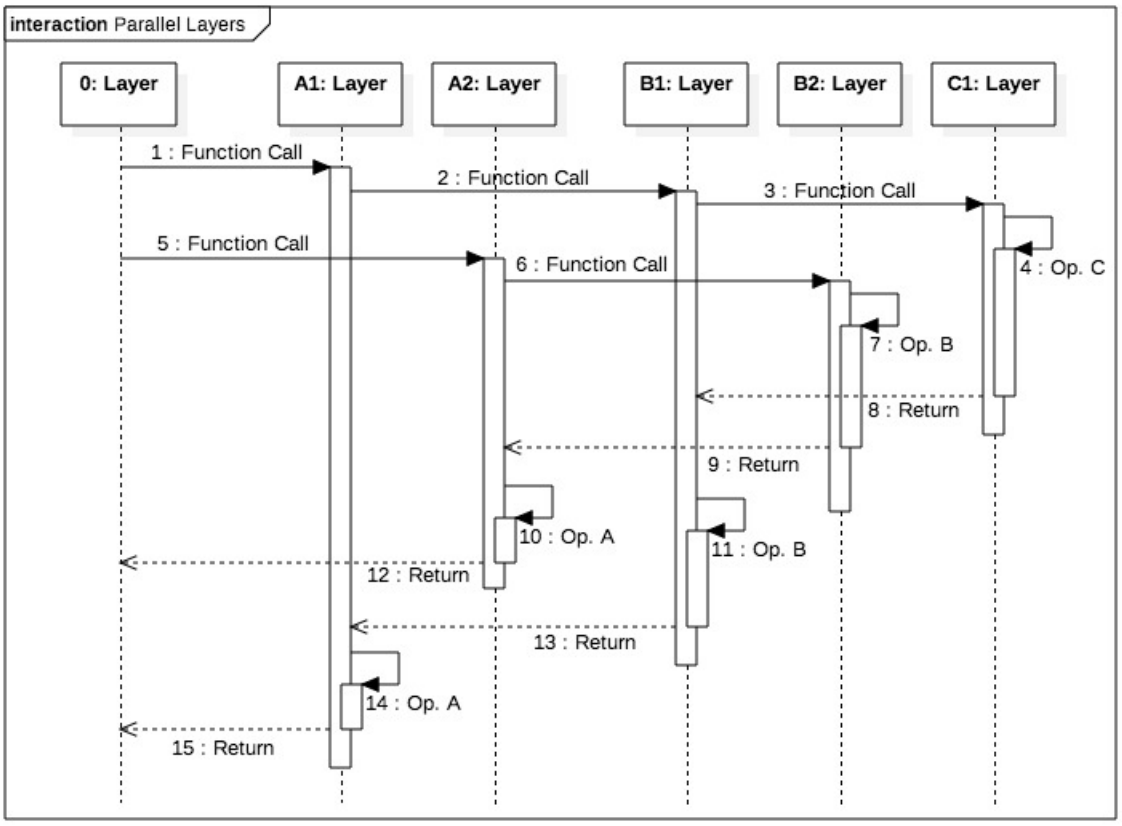

Figura 2.4: Diagrama de secuencia del patrón Parallel Layers

- La capa Layer0 llama al componente de capa A1 para realizar el cálculo 1. A1 llama al componente de capa B1 y B1 llama al componente de capa C1. Los componentes A1 y B1 esperan recibir la respuesta de sus respectivos componentes de capa inferiores por lo que permanecen bloqueados.

- El paralelismo es introducido cuando la capa Layer0 emite otra llamada para realizar el cálculo 2. El cálculo 2 no puede ser atendido por A1 por que está bloqueado. Entonces se crea otro componente de capa A2 para atender la llamada. La creación de componentes de capa puede ser de manera dinámica o estática. El componente A2 llama a otro componente de capa B2. Cada componente en cada capa realiza exactamente la misma operación. Cada componente de la capa B es capaz de atender las llamadas de cada componente de la capa A. Cada componente atiende su respectiva llamada y regresa un resultado. El objetivo principal es que todos los cálculos sean ejecutados en un tiempo más corto.

2.2.6. Ejemplo de un patrón de diseño — Multiple Local Call

El patrón Multiple Local Call [24] describe el diseño de un mecanismo de comunicación bidireccional (uno a muchos y muchos a uno) para una aplicación que implemente el patrón Parallel Layers [24]. Describe un conjunto de componentes de comunicación que son capaces de difundir peticiones a multiples componentes de comunicación que operan sobre variables globales y/o locales y devolver un resultado. Este tipo de comunicación apoya a la delegación de partes de una actividad de procesamiento a componentes de una capa inferior. A los componentes de capa inferior como superior se les permite ejecutarse de manera simultánea. Ambos tipos de componentes requieren un mecanismo de sincronización entre ellos durante cada petición. La petición es considerada de manera local por que todos los componentes están diseñados para existir y ejecutarse en una memoria compartida de un sistema paralelo. Cada componente de una capa superior debe esperar hasta que cada componente de capa inferior le proporcione un resultado.

Contexto

Un programa en paralelo se desarrolla utilizando el patrón arquitectónico de Parallel Layers [24] con una aproximación de paralelismo funcional. El algoritmo es dividido entre procesos autónomos llamados componentes de capa que forman los componentes de procesamiento del programa. El programa es desarrollado para un sistema paralelo de memoria compartida.

Problema

Una colección de capas paralelas necesita comunicarse mediante la emisión de varias peticiones de operación y esperando de manera síncrona por los resultados. Cada elemento de información está contenido en un componente de capa y solo es repartido cuando se envían a los componentes de capa inferior o son reunidos y entregados a componentes de capa superior.

Solución

La solución se puede diseñar utilizando un solo servidor multihilo. Puede recibir llamadas de un componente de capa superior. Crea varios clientes que emiten una sola llamada a componentes de capa inferior y esperan hasta que reciban la parte de la respuesta correspondiente. El procesamiento total es dividido en varios componentes de capa. Cada componente puede seguir dividiendo la parte de su procesamiento. Todos los componentes son diseñados para que puedan existir y ejecutarse al mismo tiempo en una sistema de memoria compartida. Los resultados son enviados a los componentes de capa superior después de que todas las llamadas son atendidas. Se preserva el orden preciso de las operaciones sobre los datos y el orden de los resultados.

Estructura

En la figura 2.5 se muestra un diagrama de colaboración de la estructura, participantes y sus relaciones de una sola etapa de comunicación para este patrón.

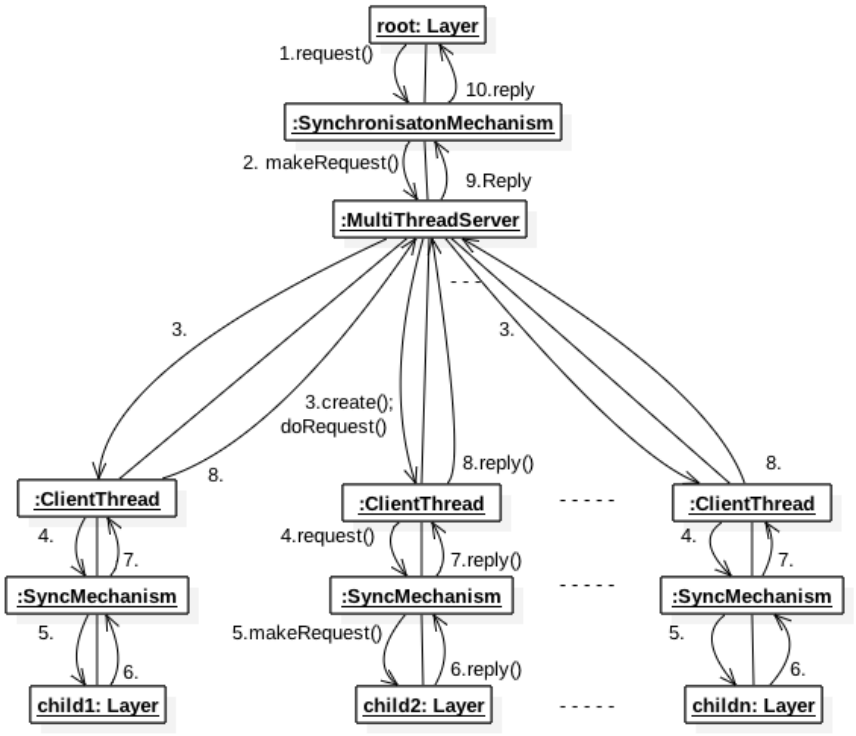

Figura 2.5: Diagrama de colaboración del patrón Multiple Local Call

Dinámica

Este patrón sirve como guía para realizar una implementación que trabaja entre componentes de diferentes capas en un sistema paralelo de memoria compartida. En la figura 2.6 se muestra un diagrama de secuencia del patrón. El escenario para ejemplificar el comportamiento incluye los siguientes pasos:

- El componente raíz produce una llamada al Multithread server mediante el componente del mecanismo de sincronización y espera por la respuesta.
- El Multithread server recibe la llamada del componente del mecanismo de sincronización y procede a crear grupos de clientes para atenderla.
- Una vez creados, a cada cliente se le envía un subconjunto de la información. Cada cliente realiza una llamada a un componente de capa inferior mediante otro componente del mecanismo de sincronización y espera por el resultado.

- Cada vez que un componente de capa inferior termina, envía sus resultados a su respectivo cliente mediante el componente del mecanismo de sincronización.
- Cada cliente envía el resultado obtenido al Multithread server. Una vez que recibió la respuesta de todos los clientes, reúne todas las respuestas en una sola y la envía al componente raíz.

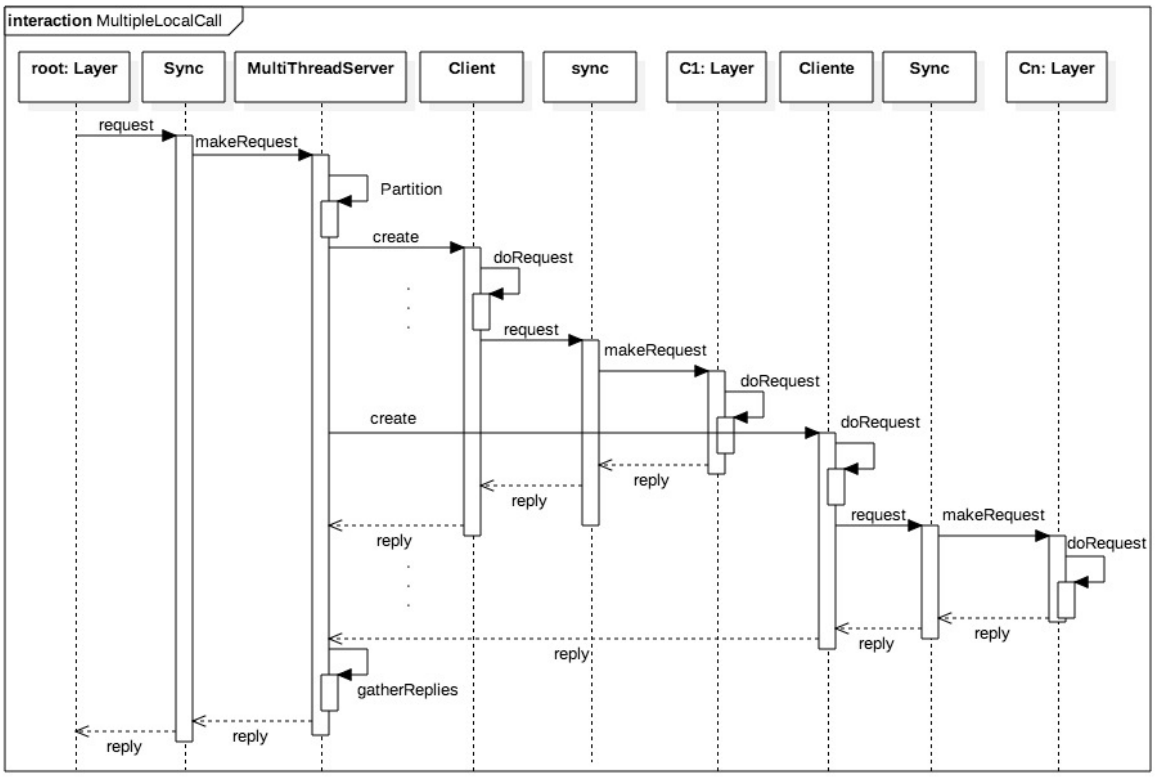

Figura 2.6: Diagrama de secuencia del patrón Multiple Local Call

2.3. Atributo de calidad de software

Un atributo de calidad de software es una propiedad que puede ser evaluada cuantitativamente para juzgar la operación de un sistema [3]. Forma parte de los requerimientos no funcionales [7]. Existen diferentes clasificaciones de los atributos

de calidad de acuerdo al autor Kai Qian [27]. Los que son de interés para este trabajo de tesis son los que se pueden observar o medir en tiempo de ejecución. En la siguiente lista se mencionan los más representativos [27]:

1. *Disponibilidad:* Es la medida de la capacidad de un sistema en la que puede ofrecer un servicio en una cantidad de tiempo, por ejemplo 24 horas 7 días a la semana. Se puede mejorar a través de la replicación de información y un diseño que considere las fallas de hardware, software, o de red.

2. *Seguridad:* Es la medida de la capacidad de un sistema para hacer frente a los ataques maliciosos fuera y/o dentro del sistema. La seguridad puede mejorarse mediante el uso de firewalls y procesos de autenticación y autorización.

3. *Desempeño:* El grado en el que un sistema realiza las funciones designadas dentro de las limitaciones dadas. Algunos ejemplos de las limitaciones son el tiempo necesario para responder a eventos específicos, el número de eventos procesados en un intervalo de tiempo y la utilización de recursos.

4. *Usabilidad:* Es el nivel de percepción de facilidad que tiene un usuario cuando interactúa con un producto o sistema. Incluye temas de integridad, exactitud, compatibilidad, una interfaz fácil de usar, documentación completa y apoyo técnico.

5. *Confiabilidad:* Es una medida de la capacidad de un sistema para seguir operando. Incluye la frecuencia entre fallas. Se puede medir con el "tiempo promedio que ocurre entre fallas" (MTTF). También incluye temas de la habilidad del sistema para recuperarse de fallas, la exactitud de los resultados de salida y la predicción de fallas.

6. *Mantenibilidad:* Mide la facilidad del sistema de software para realizarle cambios. Incluye temas de extensibilidad, adaptabilidad, utilidad, compatibilidad, facilidad de configuración y de aplicación de pruebas.

Existen parámetros por los cuales los atributos de calidad de un sistema son juzgados, especificados y medidos. A estos parámetros son conocidos como *concerns*

[3]. Un *concern* (para esta tesis) es un tema de interés que hay que considerar para juzgar el comportamiento de un sistema de acuerdo a ciertos atributos.

También existen factores de un sistema y su entorno que tienen impacto en los temas de interés. Los factores cambian de acuerdo al atributo de calidad. Los factores son propiedades internas o externas al sistema que afectan a los temas de interés [3].

El atributo de calidad a considerar en el presente trabajo de tesis es el desempeño. La naturaleza del desempeño de un sistema proviene de la satisfacción de múltiples solicitudes utilizando recursos compartidos y en la forma en como esos mismos recursos son asignados. En la figura 2.7 se muestra un diagrama de árbol de algunos temas de interés y factores del sistema mas representativos del desempeño [3]:

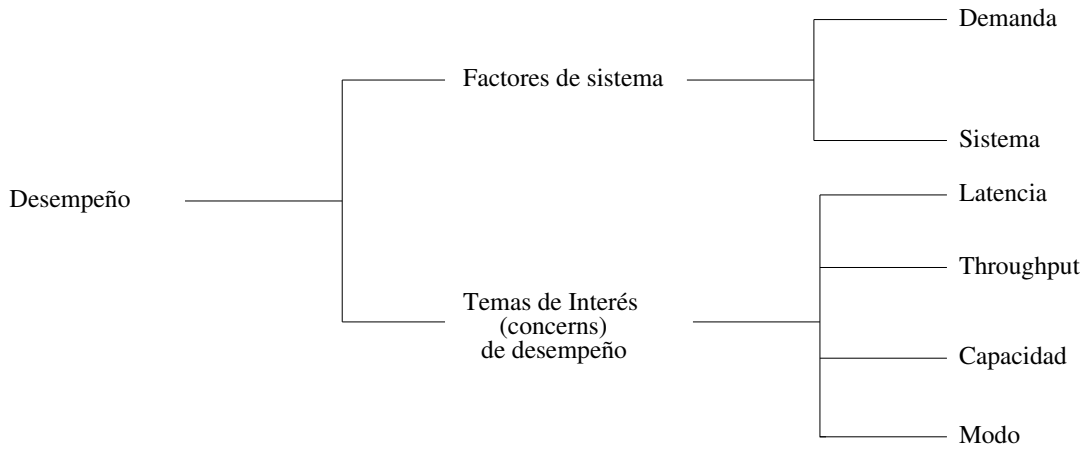

Figura 2.7: Temas de interés y factores de sistema del desempeño.

Los factores son aspectos del sistema que pueden contribuir al desempeño. Incluyen las peticiones generadas por el entorno del sistema y las respuestas del sistema a estas peticiones. A continuación se mencionan algunos de los factores de sistema representativos del desempeño [3]:

1. *Demanda:* Es una determinación de la cantidad de un recursos que se necesitan para atender las peticiones del sistema. Un ejemplo es el patrón de llegada para cada flujo de peticiones. Otro ejemplo son los requerimientos de tiempo de ejecución para atender a cada petición.

2. *Sistema:* Son los recursos de un sistema y que son necesarios para llevar a cabo las respuestas a las peticiones. Algunos ejemplos son los tipos de recursos,

servicios de software para la administración de recursos y la asignación de recursos.

Los temas de interés de desempeño son utilizados para especificar y evaluar el desempeño del sistema. Un ejemplo son los criterios para la asignación de recursos a una petición para que se ejecute en un tiempo determinado. Otro ejemplo son las limitaciones de tiempo para responder a las peticiones. A continuación se mencionan algunos de los temas de interés más representativos del desempeño [3]:

1. *Latencia:* Se refiere a un intervalo de tiempo durante el cual se debe ejecutar la respuesta a una petición. El intervalo de tiempo define una ventana de respuesta dada por un tiempo de inicio (latencia mínima) y un tiempo de finalización (latencia máxima).

2. *Throughput:* Se refiere al número de respuestas a las peticiones que se han completado sobre un intervalo de observación dado.

3. *Capacidad:* Es una medida de la cantidad de trabajo que un sistema puede llevar a cabo. Un ejemplo es el throughput máximo alcanzable de carga de trabajo en condiciones ideales. Otro ejemplo es el porcentaje de tiempo máximo alcanzable en el que un recurso está ocupado por el sistema sin dejar de cumplir los requisitos de latencia.

4. *Modo:* Un modo puede caracterizarse por la configuración de los recursos utilizados para satisfacer la demanda. Un ejemplo es que un sistema puede operar con capacidad reducida si los recursos dejan de funcionar adecuadamente. Otro ejemplo es que un sistema puede sacrificar los requerimientos de tiempo de las peticiones menos importantes durante periodos de sobrecarga.

2.4. Modelo de programación MapReduce

MapReduce es un modelo de programación y una implementación asociada para el procesamiento y generación de conjuntos de datos de volumen considerable. Está diseñado para simplificar la lógica de paralelización de tareas de una aplicación. El modelo MapReduce es una combinación de una función de mapeo y una función

de reducción [12]. La función de mapeo se aplica a cada registro de los datos de entrada y la función de reducción de aplica sobre la salida de la función de mapeo. A continuación se describe una secuencia de pasos de cómo funciona un modelo de MapReduce.

1. Un programa de MapReduce consiste en los datos de entrada, una función de mapeo y una función de reducción.

2. Los datos de entrada se dividen en porciones más pequeñas.

3. Se aplica la función de mapeo a cada porción de los datos de entrada. La salida de cada registro es una colección de pares clave-valor. Una representación de cada elemento es $< Clave_1, ValorM_1 >$ hasta $< Clave_n, ValorM_n >$

4. Se genera un nuevo conjunto de pares. Los pares con una misma clave son reorganizados en uno solo. El valor del par es una colección de elementos. La colección puede ser una estructura de datos, por ejemplo, una lista. Una posible representación para los valores con Clave 1 es $< Clave_1, Lista(ValorM_1) >$

5. Se aplica la función de reducción a cada elemento del conjunto de pares anterior. Se genera un nuevo conjunto de pares. El valor de cada par es el resultado de aplicar la función de reducción a la colección de elementos de cada clave. Una representación es $< Clave_1, ValorR_1 >$ hasta $< Clave_n, ValorR_n >$

6. La colección de pares resultado de aplicar la función de reducción es el resultado final del programa MapReduce.

En el modelo MapReduce pueden existir tantas fases de mapeo y de reducción como sean necesarias. Las funciones de mapeo y reducción pueden de ejecutarse de manera aislada [12]. Estas funciones son candidatas a ejecutarse en paralelo. Por ejemplo, un proceso de reducción puede iniciar sus tareas tan pronto como la función de mapeo haya terminado de procesar la clave correspondiente.

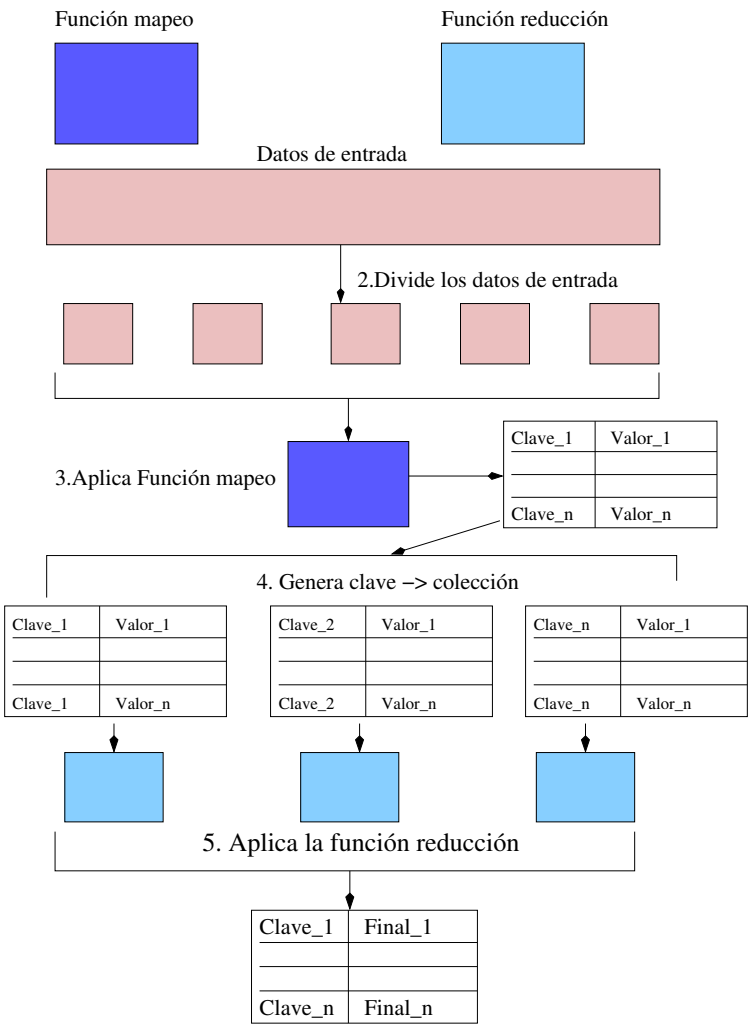

Figura 2.8: Diagrama de bloques MapReduce

2.5. Resumen.

En este capitulo se presenta una introducción a las aplicaciones Web. La manera de interactuar de los diferentes elementos que conforman una arquitectura cliente - servidor. También se presenta algunos atributos comunes que se presentan en la mayoría de las aplicaciones Web. Se menciona su entorno de aplicación. Del entorno se puede resaltar la diversidad de clientes que existen y el considerable volumen de peticiones que puede recibir una aplicación Web. Cada petición consume recursos y la cantidad recursos tiene un limite y pueden ser insuficientes para atender todas las

peticiones y afectar su desempeño.

Se realiza una introducción a la Arquitectura de Software. Se mencionan algunas definiciones de diferentes autores. Básicamente la definición para la presente tesis de la Arquitectura de Software es una descripción de la forma de distribución de sus elementos y la interacción entre ellos. Se presenta una introducción a los patrones de software. Los patrones de software son considerados como una herramienta que ha sido probada con éxito en implementaciones reales. Se describe la forma en que son descritos, su clasificación respecto a su nivel de abstracción.

A continuación, se realiza una introducción a los atributos de calidad de software. Un atributo de calidad es una propiedad deseada en un sistema. Se mencionan algunas formas de clasificarlos. El atributo de calidad de interés para esta tesis es el desempeño. Se mencionan algunos factores de interés que hay que considerar para evaluar el desempeño de un sistema y algunos aspectos internos y externos que pueden impactar en el desempeño de un sistema.

Finalmente, se describe MapReduce. MapReduce es un modelo de programación que permite simplificar la lógica de paralelización de tareas dentro de una aplicación. Se describe su funcionamiento y las diferentes fases por las cuales la información se procesa.

Capítulo 3

Trabajo Relacionado

Este capítulo presenta los trabajos relacionados al desarrollado en esta tesis. Se mencionan algunas aproximaciones relacionadas con el desempeño en aplicaciones Web [15], la relación de servicios Web REST y el modelo de computación Actor [21], y un enfoque para evaluar el desempeño de los sistemas de aplicaciones Web [18].

3.1. Aproximaciones a la construcción de aplicaciones Web de alto desempeño

Los autores Brian Goodman, Maheshwar Inampudi y James Doran, en su trabajo "Approaches to building high performance Web applications: A practical look at availibity, reliability, and performance" [15], mencionan algunas aproximaciones que ellos han adoptado enfocadas a mejorar o mantener el desempeño en aplicaciones Web como colaboradores de la empresa IBM. A continuación se mencionan cuatro de las aproximaciones propuestas por estos autores [15].

1. Caché de recursos Web. El concepto de caché de recursos Web es similar al concepto de memoria caché en un sistema. El procesador trabaja a una velocidad mayor que los sistemas de memoria. La memoria caché en un sistema es una pequeña cantidad de memoria que funciona casi a la misma velocidad del procesador. Cuando el procesador encuentra la información que necesita en la memoria caché no tiene que reducir la velocidad. Cuando no encuentra la información solicitada en la

memoria caché debe buscar la información directamente lo cual implica un costo en el desempeño.

El objetivo es almacenar recursos Web en caché con anticipación para atender peticiones en un futuro. Las diferencias con la memoria caché de un sistema son la falta de uniformidad de tamaño de los elementos utilizados en una aplicación Web, los costos de recuperar elementos del caché y la capacidad del almacenamiento. Para manejar el tamaño del objeto, las operaciones en el caché y su diseño deben de tomar en cuenta el porcentaje de peticiones que son atendidas por el caché y la tasa de bytes obtenida del caché por atender las solicitudes. El costo de recuperar un elemento del caché varía de acuerdo con el tamaño del elemento, la distancia recorrida, la congestión de la red, y la carga de trabajo presente en el servidor. Los elementos que son candidatos para almacenar en caché son aquellos que no están personalizados para algún cliente en específico o que no son actualizados constantemente.

La forma en que los autores implementan el caché en una aplicación Web es almacenando recursos Web en estructuras de datos información proveniente de un sistema de archivos o tablas de base de datos para acelerar el tiempo de transacción. En la figura 3.1 se muestra un diagrama de componentes de la implementación.

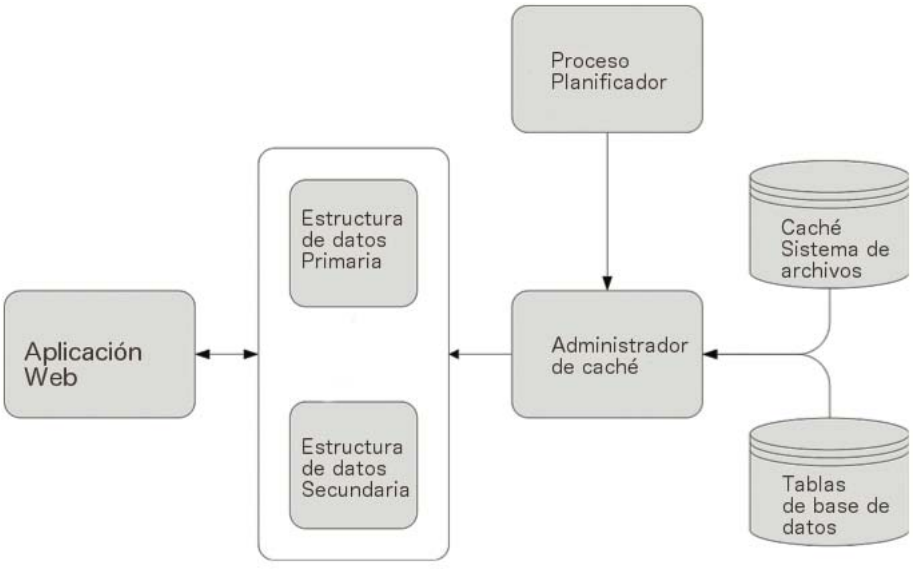

Figura 3.1: Diagrama de componentes de cache de recursos Web.

A continuación se mencionan las actividades principales de cada componente:

- Proceso planificador: Se encarga de inicializar el administrador de caché y generar los eventos para que la información del caché sea actualizada.
- El administrador de caché es responsable de realizar peticiones a las diferentes fuentes de información y cargarlas en el caché. Las fuentes pueden ser un sistema de archivos o tablas en base de datos.
- Estructura de datos: Existen dos estructuras de datos (primaria y secundaria) las cuales son utilizadas como memoria caché. La información de la segunda estructura puede ser borrada o guardada dependiendo de la estrategia de actualización. Un ejemplo de ello es información que se ha actualizado en alguna fuente de información. El objetivo es solo borrar los elementos que fueron actualizados sin necesidad de transferir nuevamente toda la otra información que no ha cambiado. Lo anterior permite que la actualización de información pueda ocurrir con frecuencia y con menos rotación de información.

2. Procesamiento asíncrono de tareas en aplicaciones Web. En esta aproximación los autores proponen eliminar o reducir las tareas que forman parte del procesamiento de una petición y que no son esenciales para la atención de la misma. Un ejemplo de ello son los eventos que se contabilizan para generar estadísticas en tiempo real. Las estadísticas en tiempo real en una aplicación Web pueden proporcionar información acerca de cuantas veces ha sido consultado cierta página, cuales fueron las ultimas diez búsquedas, etc.

Cuando se realiza una petición a una aplicación Web, la petición puede ser candidata a generar información para algunas estadísticas. La operación de recolectar y procesar información para las estadísticas también consume recursos y puede aumentar el tiempo de respuesta a una petición. El tiempo que emplean las tareas para las estadísticas puede ser reducido si se realizan en forma asíncrona.

Los autores implementan una solución basada en el patrón de diseño Blocking Queue [19]. El patrón de diseño Blocking Queue es una variante del patrón de diseño productor - consumidor. El patrón productor - consumidor coordina la producción y el consumo asíncrono de información o de objetos [16]. En el patrón Blocking

Queue, existen productores que colocan mensajes en una cola de mensajes y existen consumidores que obtienen mensajes de la cola para procesarlos. La funcionalidad del Blocking Queue es obtener y proporcionar los mensajes a los consumidores sólo cuando hay mensajes presentes. Lo anterior evita que los consumidores ocupen recursos al preguntar si existen mensajes en la cola para procesarlos. En la figura 3.2 se muestra el diagrama de clases proporcionado en el trabajo de los autores:

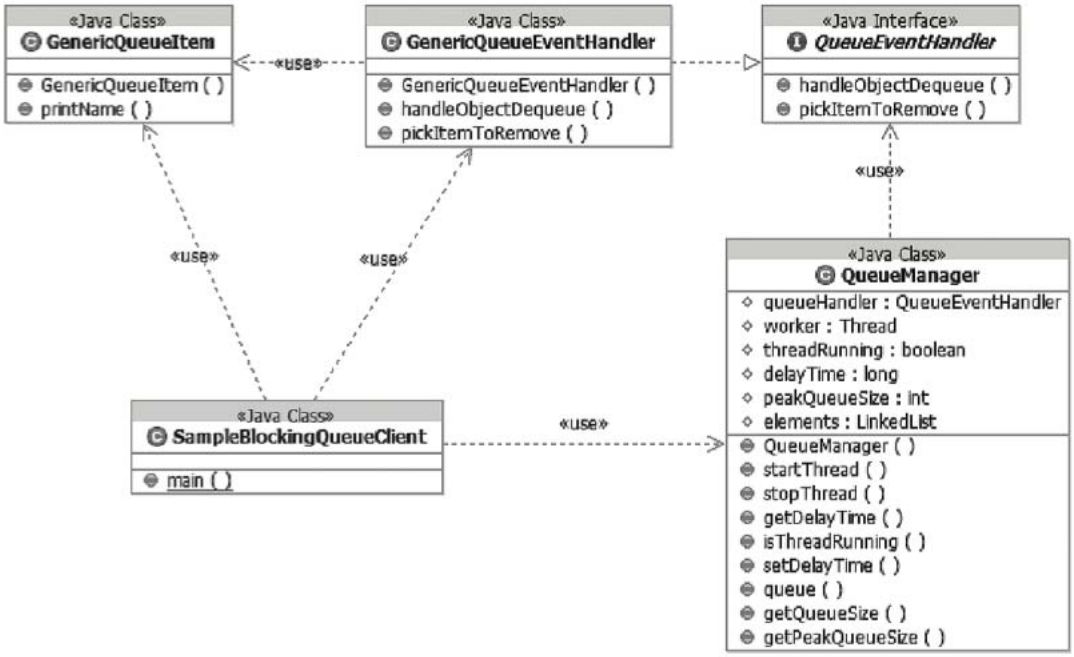

Figura 3.2: Diagrama de clases Blocking Queue.

A continuación se mencionan las responsabilidades principales de cada clase:

- *QueueClient*: Es el componente cliente del QueueManager.
- *QueueManager*: Gestiona y maneja los elementos en la cola. Contiene un buffer de elementos de tipo GenericQueueItem.
- *QueueEventHandler*: Inicializa a QueueManager y utiliza una interface para interactuar con otras implementaciones de manejo la cola.
- *GenericQueueItem*: Clase que representa a un elemento que se coloca o quita de la cola.

El ejemplo del trabajo citado es realizar una búsqueda de palabras clave en una página Web que contiene información de perfiles de empleados. Un usuario puede buscar empleados de una empresa que tienen conocimiento de un determinado cliente y una tecnología. La página Web procesa la petición de manera asíncrona agregando a la cola un objeto con algún valor (por ejemplo, las palabras clave para realizar la consulta, el ID de usuario, etc.) y después presenta los resultados de la petición. Para atender la petición se utiliza un objeto QueueEventHandler que decide el elemento a obtener de la cola y procesarlo. El procesamiento puede ser alguna búsqueda, agrupación, etc. y el resultado se almacena para después mostrarlo al cliente.

3.Comportamiento autónomo autosuficiente. En esta aproximación los autores proponen integrar a un sistema (hardware y software) un enfoque proactivo dirigido a mantener un nivel de desempeño y confiabilidad con poca o ninguna intervención humana. El trabajo de los autores está enfocado a aplicaciones que realizan a peticiones a interfaces externas. Algunos ejemplos son Servicios Web o bases de datos. El desempeño de las aplicaciones depende del funcionamiento de las interfaces externas. En caso que las interfaces tengan un mal desempeño o no estén disponibles potencialmente afectan a todas las aplicaciones que dependen de ellas. Se propone una práctica para que las aplicaciones puedan emplear un enfoque autónomo y resistente a los posibles fallos de las interfaces externas que utilizan.

Los errores de peticiones a interfaces externas se pueden detectar mediante algún mensaje de error. Si cuando un error o un comportamiento anormal no se informa, puede provocar que se desperdicien recursos. Un ejemplo de ello es la espera de la respuesta a una petición a una interfaz externa o que nunca se obtenga una respuesta satisfactoria. Con el objetivo de proporcionar un mayor nivel de resistencia, se puede estar preguntando por el estado de manera recurrente. Los Servidores UDDI (Universal Description, Discovery, and Integration) pueden identificar varios puntos finales. El cliente puede utilizar diferentes servicios Web para realizar la misma tarea de acuerdo a su estado. El sistema de disponibilidad de servicio incluye componentes como el consumidor de servicios, el componente que se encarga de localizar los servicios, el proveedor de servicios y el componentes que se encarga de la interacción entre ellos. En la figura 3.3 se muestra un diagrama de bloques del sistema de

disponibilidad de servicio.

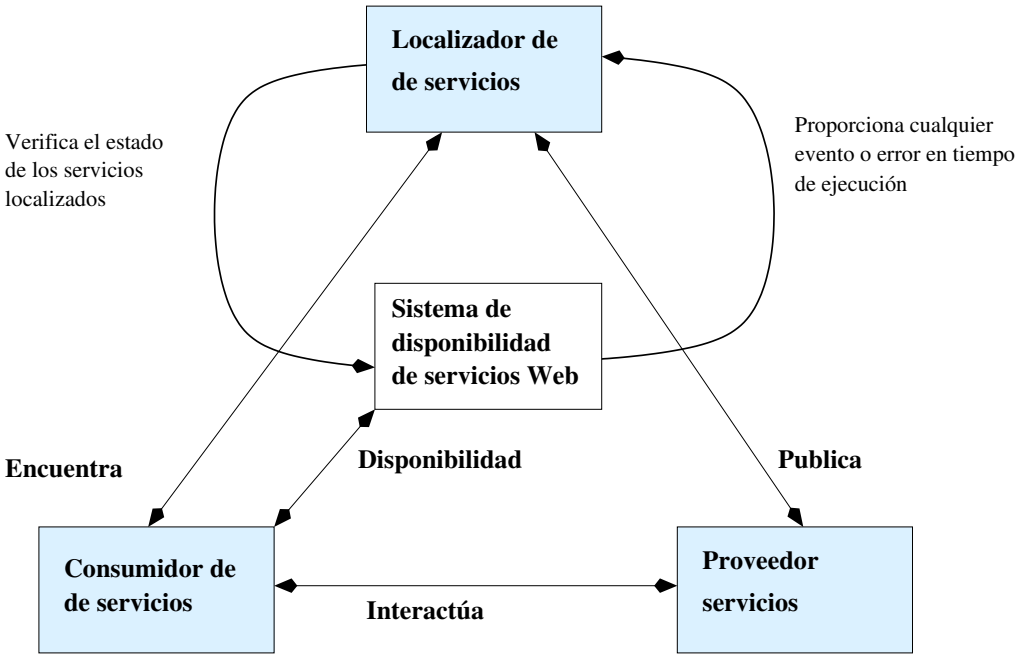

Figura 3.3: Diagrama de bloques del sistema de disponibilidad del servicio .

Los consumidores de los servicios pueden ser advertidos de ciertos problemas comunicándose con un tercer componente independiente. El consumidor puede mantener un mayor nivel de servicio implementando la lógica apropiada de que servicio utilizar. La respuesta obtenida por el consumidor del servicio es independiente de la petición realizada al proveedor de servicios. Las actualizaciones del estado de la disponibilidad se pueden hacer mediante otras herramientas de supervisión. Lo anterior permite una base de conocimiento de la disponibilidad del servicio integrado.

4.Degradación elegante. El objetivo es mantener a los usuarios con la sensación de que el sistema siempre está funcionando. La clave es evitar que características no disponibles afecten a otras que sí pueden proporcionar cierto servicio. Para lograr la degradación se puede modificar la experiencia del usuario y comunicarle que algunas características no están disponibles. De esta manera se puede mantener el desempeño y disponibilidad de la aplicación hasta que recupere.

Las aplicaciones Web son vulnerables a un número creciente de dependencias:

servicios Web, bases de datos, sistemas de archivos, etc. Cualquier dependencia puede provocar que la aplicación Web se detenga. Un método para manejar este tipo de situaciones es establecer un tiempo de espera en la transacción. Con el tiempo de espera se puede terminar con el proceso y devolver algún error al usuario final. Los autores promueven que es mejor notificar de un error que exhibir que realmente no está disponible.

El siguiente paso consiste en impedir el acceso futuro al recurso que tiene el fallo hasta que esté nuevamente disponible. El desafío es monitorear la disponibilidad de esos recursos. Un ejemplo de ello es un fallo en una base de datos. Una copia de seguridad puede ser utilizada o desactivar algunos elementos de menú. Indicando al usuario final que algunas de las características se han desactivado temporalmente.

En esta aproximación los autores solo mencionan los objetivos que se deben de alcanzar. Uno de los objetivos es que cuando una aplicación comienza a fallar, afecta a los usuarios finales u otros sistemas. Si una sola funcionalidad está esta fallando, se pueden desactivar los componentes que dependen de esa característica y proporcionar un método para reanudar el trabajo. En caso que de no se pueda recuperar del error, se puede proporcionar un método para continuar trabajando con alguna copia de seguridad.

3.2. Servicios Web REST y el modelo de computación Actor

En su trabajo "On Actors and the REST" [21] los autores Janne Kuuskeri y Tuomas Turto investigan la relación entre el modelo de computación Actor y el estilo de Arquitectura de Software de transferencia del estado representacional (Representational State Transfer o REST) para sistemas distribuidos [21]. Se muestra cómo un modelo de Actor restringido puede utilizarse para representar los principios subyacentes de REST. También los autores sugieren una notación para describir sistemas REST. En su trabajo los autores consideran las interfaces REST sólo en el contexto de los servicios Web.

3.2.1. El Modelo de Actor

Hewitt et al. [17] propusieron inicialmente el modelo de computación Actor para de mejorar los lenguajes de programación para Inteligencia Artificial [21]. En el trabajo de Janne Kuuskeri y Tuomas Turto se describe a los Actores como un modelo de computación concurrente en sistemas distribuidos. A continuación se mencionan algunas características del modelo de Actor consideradas por estos autores[21]:

Actor. El modelo de computación Actor es un sistema compuesto de una multitud de agentes computacionales. Cada agente es un Actor. Un Actor es una entidad activa y autocontenida que es identificada por su dirección. Los Actores encapsulan toda la información necesaria, según lo especificado por el comportamiento de los Actores, para que pueda funcionar como parte del sistema. Los Actores no comparten ningún estado y se comunican sólo por el paso de mensajes [21].

Existe un buzón conceptual para cada dirección que agrega a una cola los mensajes entrantes. En el modelo de Actor se garantiza que los mensajes enviados van a ser eventualmente entregados. No se garantiza el tiempo que se tarda en entregar un mensaje. Tampoco específica el orden en el que van a llegar al buzón del Actor destino. No es posible enviar un mensaje a un Actor de forma arbitraria sin conocer primero su dirección.

Comportamiento operacional. La manera en que un sistema de Actores funciona es a través del envío y recepción de mensajes. A continuación se mencionan las acciones que un Actor puede realizar cada vez que recibe un mensaje [21] :

- Decidir sobre su comportamiento de reemplazo. Esencialmente es otro Actor que toma el lugar del Actor que lo crea, con el fin de responder a ciertas comunicaciones.
- Puede crear nuevos Actores.
- Pueden enviar mensajes a otros Actores.

Las tres acciones pueden ocurrir al mismo tiempo. Una vez que el comportamiento de reemplazo se ha decidido, otro Actor es creado para representar el Actor original. El nuevo Actor es capaz de procesar el siguiente mensaje del buzón.

3.2.2. Actores y REST

En la tabla 3.1 se muestra una correspondencia establecida por Kuuskeri y Turto entre los conceptos del modelo de Actor y REST [21]:

Tabla 3.1: Correspondencia entre los conceptos de Actor y REST

Modelo de Actor	***REST***
Actor	Recurso
Nombre de buzón	URL
Actores conocidos	Link de Hypertexto
Mensaje	Petición HTTP
Comportamiento	Cambio de estado del recurso
Cliente	Consumidor

A continuación se mencionan cada una de las comparaciones [21]:

Actor y Recurso. Ambos están destinados a denotar una entidad autónoma y aislada. Son los componentes básicos fundamentales para su respectivo enfoque. La interfaz externa del sistema construido se define en términos de Actores o recursos. En los servicios REST los recursos disponibles definen el vocabulario para el servicio Web. En un sistema de Actores, los Actores externos determinan las entidades a las cuales se le pueden enviar mensajes.

Buzón y URL. Los Actores son identificados por sus direcciones de buzón. En un diseño REST se utiliza direcciones URLs para denotar los recursos que expone un servicio. En ambos escenarios las direcciones son la única manera en la que un cliente externo puede interactuar con el sistema. Ambos enfoques permiten a las entidades reenviar los mensajes recibidos a otras direcciones.

Actores conocidos y link de hypertexto. En el modelo de Actor un nodo sólo puede comunicarse con otro nodo si tiene su dirección. A un servicio REST solo se le puede hacer una petición si se conoce su dirección URL. Un conjunto de direcciones externas se declaran cuando se inicia el sistema. Los Actores o servicios REST que tienen las direcciones externas comprenden la interfaz visible del sistema. Los Actores o servicios con interfaz externa pueden comunicarse

con otros elementos que sólo pueden ser utilizados a través de la interfaz visible al exterior del sistema.

Comportamiento y cambio de estado de recurso. En el modelo de Actor, la funcionalidad de un Actor se define por su comportamiento. El comportamiento cambia con el tiempo conforme el Actor va procesando los mensajes y decide los nuevos comportamientos de reemplazo. En REST, un recurso tiene estado. El estado cambia con el tiempo conforme se van procesando las solicitudes.

Cliente y Consumidor. El término cliente se refiere al remitente del mensaje en el modelo de Actor. En REST el consumidor se refiere a la entidad que realiza una petición. Un ejemplo de ello es el navegador Web. En el modelo de Actor un "valor de retorno" de un comportamiento significa enviar un mensaje al cliente. En un servicio REST un mensaje de respuesta siempre sucede después de un mensaje de una petición. La diferencia es que en el modelo de Actor el mensaje de respuesta no es obligatorio.

3.2.3. Ejemplo

El siguiente ejemplo es planteado por los autores para presentar sus ideas. Es un servicio simple para crear entradas y relaciones en un blog sin tomar en cuenta cuestiones de autenticación, comentarios, etc. Primero se considera el servicio como un servicio Web REST. Posteriormente se considera el servicio como una red de Actores que procesan mensajes HTTP.

Tanto en REST como en el modelo de Actor, se decide sobre los recursos y después sobre el comportamiento. Para el servicio REST, el recurso raíz es el contenedor de todas las entradas del blog. La dirección asignada para el ejemplo es /blog. Para el modelo de Actor significa que existe una dirección de un buzón /blog. Las nuevas entradas agregadas se convierten en nuevos recursos. Los nuevos recursos tienen una dirección. Un ejemplo de una dirección es /blog/2010-01-10. Para el modelo de Actores significa que nuevos Actores son creados para manejar los mensajes de las nuevas direcciones de buzón.

En la tabla 3.2 se muestra una relación entre los mensajes soportados, los métodos y comportamientos asociados [21] :

Tabla 3.2: Mensajes soportados y comportamientos asociados

Método/Dirección	***/blog***	***/blog/2010-01-10***
POST	Crea un Actor con la dirección /blog/2010-01-10 con el comportamiento por defecto, envía código 201 al cliente	Sin efecto, envía código 405 al cliente
GET	Envía un código 200 al cliente con todas direcciones (Actores) de los blog disponibles	Envía un código 200 al cliente con el contenido del blog
PUT	Envía un código 405 al cliente sin tener efecto	Cambia el comportamiento del Actor para reflejar el contenido actualizado. Enviar un código 200 al cliente
DELETE	Envía un código 405 al cliente sin tener efecto	Todos los mensajes envían código 404 después de devolver un código 200 a cliente

Los códigos numéricos de la tabla 3.2 son de la especificación de HTTP :

Tabla 3.3: Códigos Númericos HTTP utilizados

200	Petición exitosa.
201	Creado.
404	Recurso no encontrado.
405	Método no permitido.

Para examinar mejor la relación entre el modelo de Actor y REST los autores utilizan una notación basada en la notación de Agha [1] para la definición de los servicios Web REST basada en Actores.

— blog[list] =

— **GET** →

— send 200[self/latest] to customer

```
—        PUT →
—                send 405 to customer
—        POST[c] →
—                new blogentry(call POST[c] to list) @ self/current-date
—                send 201[self/current-date] to customer
—        DELETE →
—                send 405 to customer
— blogentry[item] =
—        GET →
—                send 200[call GET to item] to customer
—        PUT[c] →
—                send PUT[c] to item
—                send 200 to customer
—        POST →
—                send 405 to customer
—        DELETE →
—                send DELETE to item
—                send 200 to customer
—                become 404
```

La forma en como se inicia el sistema es con la instrucción

```
—                new blog(new list) @ /blog
```

La instrucción crea un nuevo blog y lo vincula con la dirección blog. A continuación se menciona la explicación por parte de los autores del diseño:

Se definen dos Actores de comportamiento. El Actor blog contiene el comportamiento para todas las entradas del blog. El Actor blogentry contiene el comportamiento de una sola entrada del blog. Los autores omiten por cuestiones de brevedad el código de los Actores item y list. El Actor item representa una entrada del blog y el Actor list una lista de entradas. Existen nuevas notaciones agregadas por los autores [21] :

- self se refiere a la dirección propia de una instancia del Actor.
- El Símbolo @ realiza un vínculo entre un Actor y una dirección dada.

- La operación call es similar a send pero se realiza de forma síncrona.

La operación blog.GET: La instrucción self/latest es la dirección de la última entrada de blog del Actor blog. Los mensajes enviados a esta dirección se dirigen al buzón de correo de diferentes Actores que contienen en su dirección la fecha en que fue creado.

La operación blog.POST: La operación call POST [c] to list envía un mensaje POST al Actor list para agregar un item c y espera de manera síncrona por él. Cuando devuelve el valor se crea un nuevo Actor blogentry. El item devuelto de la lista es proporcionado como un Actor conocido al nuevo Actor blogentry y se vincula con su propia dirección a la cual se le agrega la fecha actual. Los métodos delete y put operan de manera similar.

3.3. Un enfoque para evaluar el desempeño de los sistemas de aplicaciones Web

Tahani Hussain, en su trabajo "An Approach to Evaluate the Performance of Web Application Systems" [18] propone un enfoque para evaluar el desempeño del sistema de aplicación Web. Consiste en tres modelos implementados en el lenguaje Java: el modelo de carga de trabajo, el modelo de simulación y el modelo de análisis de desempeño. A continuación se mencionan cada uno de los modelos.

3.3.1. Modelo de carga de trabajo

Consiste en dos componentes: registrar y generar. Cada componente se aplica a ambientes de trabajo diferentes. El componente de registrar se aplica a un ambiente de trabajo real y el componente de generar se aplica en un ambiente de simulación.

El componente de registrar escribe en un archivo la secuencia de las peticiones a una aplicación Web generadas por los usuarios en un entorno real de trabajo. Un componente se encarga de recibir la petición, registrar y después reenviarla a su destino original. Se instala entre el usuario y la aplicación Web. A continuación se mencionan los pasos que se realizan en una petición de usuario.

Paso 1. El cliente Web realiza una petición a la aplicación que primero es recibida por el componente de registrar y reenviar.

Paso 2. El componente recibe la petición, realiza el registro de la petición en un archivo de salida y reenvía la petición a la aplicación Web.

Paso 3. La aplicación Web recibe la petición y se encarga de atenderla.

Paso 4. La respuesta es recibida nuevamente por el componente de registrar y reenviar.

Paso 5. Se le reenvía la respuesta de la petición al cliente Web.

En la figura 3.4 se muestra un diagrama de secuencia de la interacción de los componentes modelo.

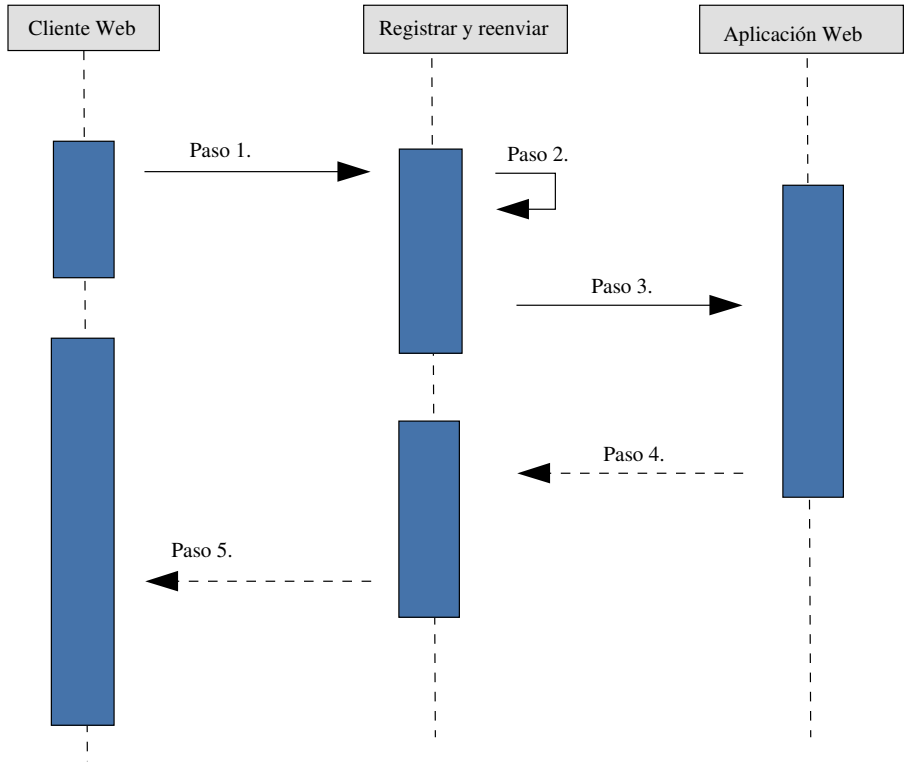

Figura 3.4: Diagrama de componentes del modelo de carga de trabajo.

El archivo de salida de texto se utiliza como entrada para replicar el comportamiento en un ambiente de trabajo similar al ambiente de trabajo real.

3.3.2. Modelo de simulación

El objetivo del modelo de simulación es reproducir escenarios basados en el modelo de carga de trabajo en una aplicación Web para registrar algunas mediciones de desempeño consideradas para el análisis. Las métricas consideradas para el desempeño de una aplicación Web son el tiempo promedio de respuesta, el número total de peticiones HTTP y la carga de trabajo.

3.3.3. Modelo de análisis de desempeño

El objetivo principal del modelo es realizar cálculos sobre la información obtenida del modelo de simulación. Basado en los cálculos se puede determinar el nivel de satisfacción del usuario final, identificar cuellos de botella del sistema y comprobar la eficiencia del sistema.

Métricas de desempeño.
Tahani Hussain [18] considera como métricas principales el tiempo de respuesta y el número total de peticiones HTTP realizadas a la aplicación Web. El tiempo de respuesta se define como el tiempo en segundos que tarda el sistema de aplicaciones Web para responder a las peticiones de los usuarios [18]. Es determinado desde el momento de enviar el primer byte de la petición del usuario ($\mathbf{TF_U}$) y el momento de recibir el primer byte de la respuesta de la aplicación Web ($\mathbf{TF_S}$). De acuerdo a la autora, el tiempo medio de respuesta (R) se calcula utilizando la ecuación 3.1:

$$R = \frac{\sum_{i=1}^{N} \mathbf{TF_{S}}\boldsymbol{i} - \mathbf{TF_{U}}\boldsymbol{i}}{N} \tag{3.1}$$

Donde N es el número total de peticiones atendidas con éxito. La métrica de la desviación estándar σ es un verificador de control de calidad. La ecuación 3.2 se usa para calcular σ.

$$\sigma = \frac{\sum_{i=1}^{N}((\mathbf{TF_{S}}\boldsymbol{i} - \mathbf{TF_{U}}\boldsymbol{i}) - R)^2}{N} X 100\,\% \tag{3.2}$$

La autora define que si $0.1 < \mathrm{R} \leq 2$ segundos es un tiempo de respuesta deseable. Si $2 \leq \mathrm{R} \leq 5$ segundos es un tiempo de respuesta aceptable. Un tiempo de respuesta $\mathrm{R} > 5$ no es considerado aceptable.

Si σ es $\leq 33\,\%$, entonces el sistema funciona de manera constante si la carga de

trabajo aumenta. De lo contrario, el sistema es afectado de forma considerable por el aumento de la carga de trabajo.

Número total de peticiones HTTP

El número total de peticiones HTTP (NR) se mide contando todas las peticiones realizadas por todos los usuarios a la aplicaciones Web HN_i:

$$NR = \sum_{i=1}^{X} HN_i \tag{3.3}$$

Si NR $\leq$ 2N, el diseño de la aplicación Web no es aceptable. De lo contrario, el diseño del sistema es aceptable.

Throughput

Para el sistema de aplicación Web, el throughput (TH) se puede definir como el número promedio de peticiones atendidas por el sistema por unidad de tiempo [18]. En el trabajo citado se propone la ecuación 3.4:

$$TH = \frac{N_s}{T_s} \tag{3.4}$$

Donde T_s representa el tiempo total que tarda el servidor para atender un número de solicitudes N_s expresado en segundos. TH se expresa como solicitudes por segundo o RPS. En el trabajo de Tahani Hussain, se considera que si TH $<$ 5 RPS el throughput es inaceptable. De lo contrario, el throughput del sistema es aceptable. Si existen varios servidores y los throughput no son iguales entre sí con una variante más del 15 %, se observa por separado la capacidad de cada servidores.

3.3.4. Ejemplo

La autora plantea dos escenarios de ejemplo. La configuración de simulación para el Sistema A consta de 32 usuarios, infraestructura de Internet, un cortafuegos, un router, y un switch. El servidor Web necesita un total de 395.500 milisegundos de tiempo de procesamiento junto con el tiempo de espera para cada petición. Los retrasos de colas para los nodos de Internet, cortafuegos, router y el switch, con base en la carga de trabajo generada escenario del modelo anterior, son 1.2, 0.51, 0.25 y 0.38 milisegundos respectivamente. El modelo de la configuración del Sistema A se

muestra en la figura 3.5.

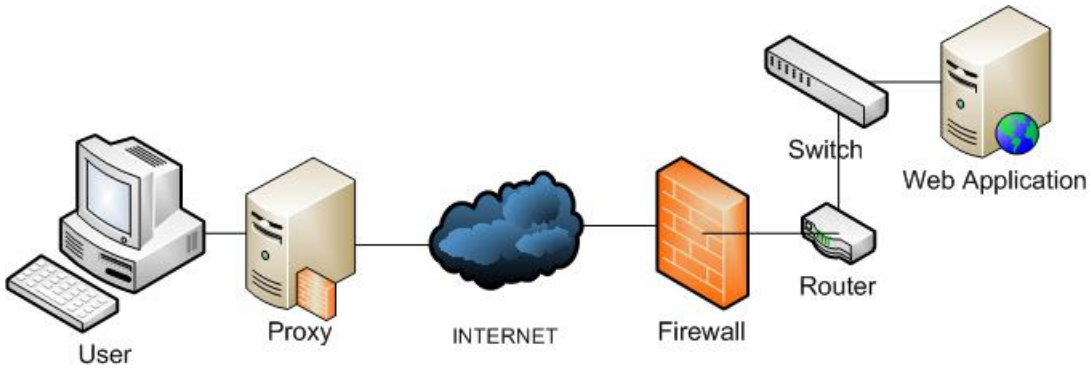

Figura 3.5: Modelo del Sistema A.

Desde el resultado de la simulación, se encuentra que el Sistema A requiere alrededor de 1,4 horas de simulación, y sirve para procesar y 528 peticiones de los 32 usuarios virtuales. Los resultados se ilustran en la Tabla 3.4.

Tabla 3.4: Resultados de desempeño del Sistema A

Metrica	Valor	Observación
R	7.63	Tiempo de respuesta no aceptable
σ	76.9 %	El sistema es inestable con la carga de trabajo
NR	1848	Diseño del sistema no aceptable (NR=3.5N)
TH	2.1	throughput no aceptable

La interpretación de los resultados es la siguiente: existe sobrecarga de peticiones HTTP (NR). Cada petición necesita en promedio el tiempo de 3.5 solicitudes HTTP para que sea atendida. TH es 2.1, lo que significa que el 82 % del tiempo de simulación se desperdicia en los costos generales del tráfico HTTP que viaja por la red, mientras que el 18 % del tiempo de la simulación se ocupa para el procesamiento de las solicitudes. Por último, σ y R son altos; lo que significa que la mayor parte de los tiempos de respuesta se expanden dentro de un amplio rango de valores.

La configuración de la simulación para el Sistema B se compone de 50 usuarios, la infraestructura de Internet, cortafuegos, router, switch, y los servidores W (Servidor Web), A (Servidor de aplicaciones) y D (Servidor de base de datos). Los Servidores W, A y D necesitan 53.900, 218.900 y 971.400 milisegundos respectivamente de tiempo de procesamiento. Los retrasos de colas para los nodos de interconexión son las mismos que en el primer experimento. La configuración del modelo para el sistema B se muestra en la figura 3.6.

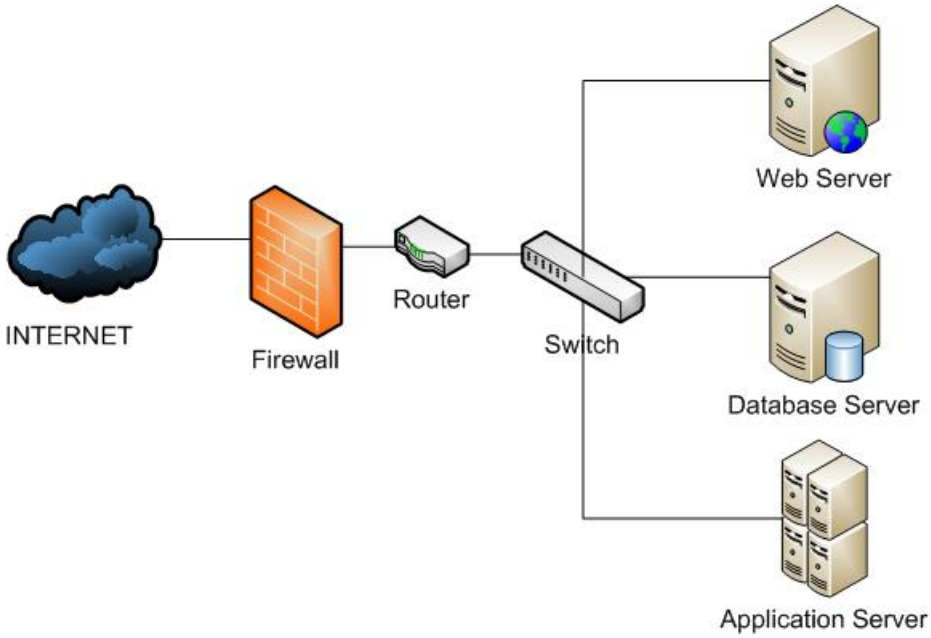

Figura 3.6: Modelo del Sistema B.

Los resultados en el Sistema B se encuentra que requiere 2.3 horas de simulación para procesar y sirve 1997 peticiones de 50 usuarios virtuales. En la tabla 3.5 se muestran los resultados del analizador de desempeño. El Sistema B es un mejor diseño en cierta medida, R $<$ 5 segundos. Es un sistema estable con el aumento de la carga de trabajo (σ $<$33 %) y no hay gastos generales con tráfico HTTP (NR $<$2N). De la Tabla 3.5, se observa que la capacidad de W no coincide con la capacidad de los otros dos servidores, y existe una diferencia entre ellos.

Tabla 3.5: Resultados de desempeño del Sistema B

Metrica	Valor	Observación
R	3.6	Tiempo de respuesta aceptable
σ	28 %	El sistema es estable con la carga de trabajo
NR	3295	Diseño del sistema aceptable (NR=1.65N)
TH	4.2	throughput no aceptable
TH_W	3.6	throughput no aceptable
TH_A	4.4	throughput no aceptable
TH_D	4.7	throughput no aceptable

3.4. Resumen

En este capítulo se presenta una revisión de tres trabajos relacionados con el desempeño de aplicaciones Web. En el primer trabajo relacionado menciona algunas aproximaciones enfocadas a construir aplicaciones Web de alto desempeño. Las aproximaciones descritas son: (1) Cache de recursos Web , (2) Procesamiento asíncrono (3) Comportamiento autónomo autosuficiente (4) Degradación elegante. Parte del objetivo de esta tesis es implementar algunas de estas aproximaciones proponiendo un lenguaje de patrones.

En el segundo trabajo relacionado se analiza la equivalencia de conceptos entre los servicios Web y el modelo de Actores. El trabajo establece las bases para realizar una aplicación Web mediante un conjunto de Actores. Algunas de sus características pueden apoyar a implementar algunas de las aproximaciones mencionadas en el primer trabajo relacionado.

Por último, el tercer trabajo relacionado establece un enfoque para medir el desempeño de una aplicación Web considerando algunas métricas como el tiempo de respuesta o el número total de peticiones HTTP. Este enfoque se plantea para evaluar la propuesta de esta tesis.

Capítulo 4

Arquitectura Web con procesamiento en paralelo

El objetivo principal de este capítulo es describir una Arquitectura de Software de una aplicación Web que permite dividir el trabajo que se realiza en la atención de peticiones en tareas más simples y algunas de ellas pueden ejecutarse de manera simultánea. En la primer sección se describe una Arquitectura de Software para una aplicación Web mediante el patrón arquitectónico de Layers[7]. La arquitectura descrita en la primera sección no divide el trabajo a realizar para atender peticiones. En la siguiente sección se describe una Arquitectura de Software mediante el patrón arquitectónico Parallel Layers [24] que permite dividir el trabajo en tareas más simples que se pueden ejecutar de manera simultánea. Finalmente se justifican las decisiones de diseño de cada uno de los componentes y un escenario de uso donde se puede implementar.

4.1. Diseño de una aplicación Web utilizando el patrón Layers

Una aplicación Web se puede diseñar utilizando el patrón arquitectónico Layers [7]. Se pueden utilizar las capas para dividir de forma lógica la aplicación, donde cada capa tiene componentes con responsabilidades similares y son mutuamente dependientes. Puede utilizar un repositorio de datos persistente para almacenar el estado

de la aplicación y manejar la concurrencia. La Arquitectura de Software aquí propuesta es para una aplicación Web que expone servicios REST. Las consideraciones son las siguientes:

Capa de servicios. Es la capa encarga de exponer los servicios REST. Recibe y procesa las peticiones realizadas por los clientes. Delega la funcionalidad a su capa inmediata inferior y espera el resultado para atender la petición. También es responsable de enviar la respuesta una vez obtenido el resultado.

Capa de negocios. Esta capa es responsable de realizar las operaciones de lógica de negocio del Sistema. Debe devolver el resultado obtenido a la capa de servicios.

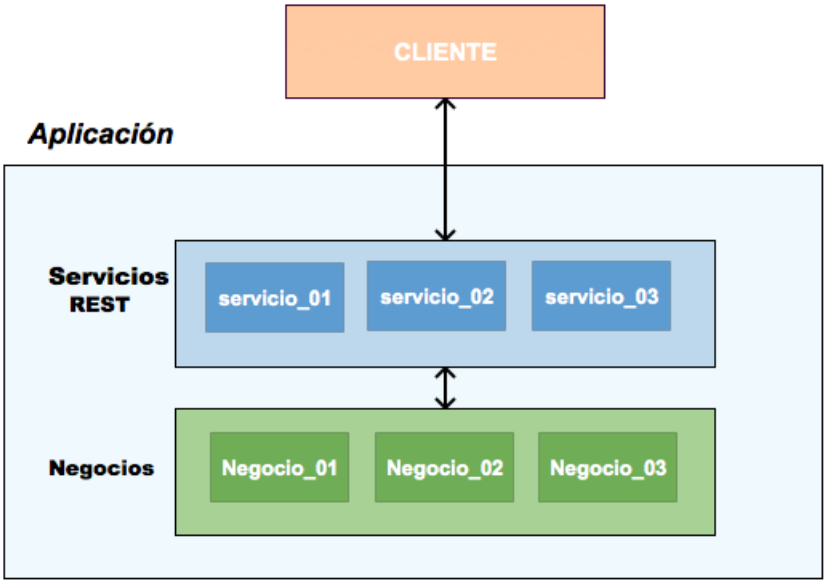

Figura 4.1: Diagrama de bloques una arquitectura para aplicaciones Web.

La ejecución de tareas de la solución aquí propuesta es secuencial. A continuación se mencionan algunos retos en el diseño de una aplicación Web con el patrón arquitectónico Layers [7]:

- La comunicación entre capas es síncrona y existe un alto acoplamiento entre ellas. Esto puede dificultar su escalabilidad y la adaptación a su entorno.

- Pueden existir operaciones que toman mucho tiempo en su ejecución y generan bloqueos. Lo anterior puede provocar desperdicio de recursos y/o que se dejen de atender otras peticiones.
- Los errores o fallas son difíciles de aislar y afectan a todo el sistema.
- Las modificaciones pueden ser transversales en todo el sistema.

4.2. Diseño de una aplicación Web utilizando el patrón Parallel Layers

Una aplicación Web también puede ser diseñada con el patrón arquitectónico Parallel Layers [24]. El patrón permite dividir el trabajo a realizar en tareas más simples. Se pueden ejecutar componentes que realizan tareas similares en paralelo. Pueden existir funciones utilizadas para atender una petición en una aplicación Web que son candidatas al paralelismo funcional. El patrón arquitectónico Parallel Layers [24] puede diseñarse con actores como componentes de capa. Las consideraciones de diseño de una aplicación Web que utiliza el modelo de actores para realizar procesamiento en paralelo son las siguientes:

- La unidad básica del procesamiento son los actores. Son componentes independientes identificados con una dirección que pueden comunicarse con otros actores mediante mensajes.
- Los actores pueden crear otros actores, administrarlos y coordinarlos. Fomenta una descomposición lógica de una tarea en partes más pequeñas que pueden ser delegadas a otros actores y ejecutarse en paralelo. Se puede construir una estructura jerárquica en forma de árbol.
- Una estructura jerárquica se puede diseñar con el patrón arquitectónico Parallel Layers. Cada capa del patrón puede ser vista como un conjunto de actores que ejecutan la misma acción.
- Con la naturaleza de los actores se puede implementar mecanismos de comunicación asíncrona o síncrona entre las capas.

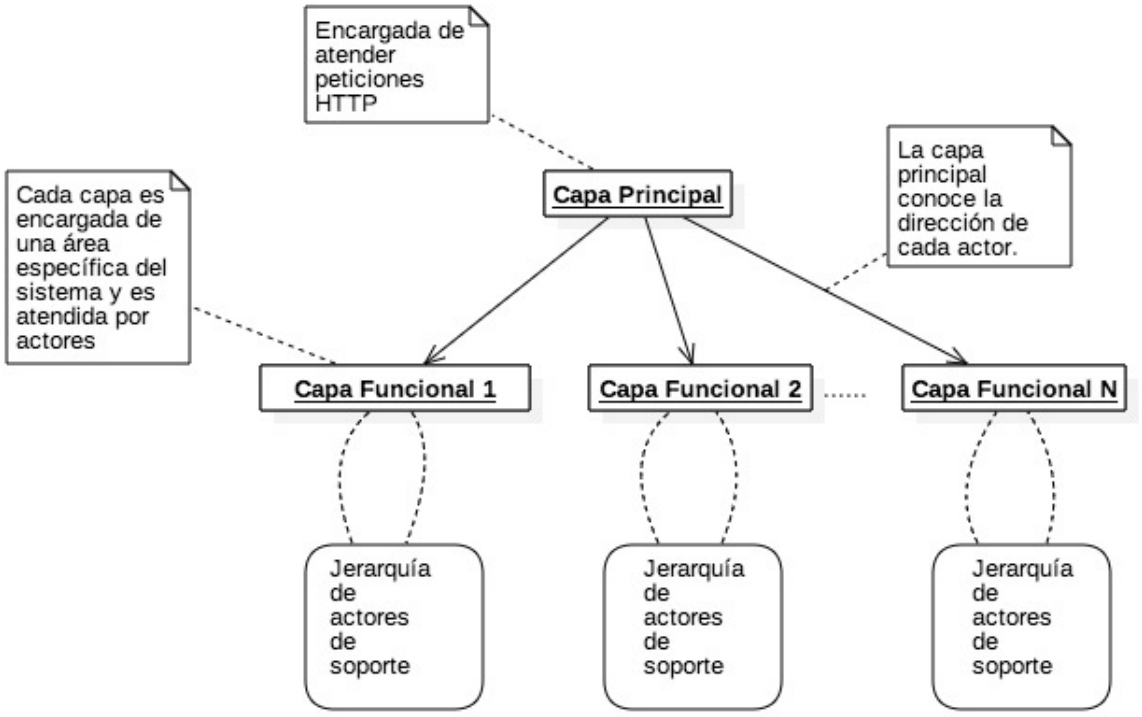

Figura 4.2: Diagrama de bloques de Parallel Layers con actores

4.2.1. Escenario de aplicación

Modelo de programación Map Reduce.

Una aplicación Web se puede describir mediante el patrón arquitectónico Parallel Layers [24] que implemente el modelo de programación MapReduce. Las unidades básicas computacionales son los actores, los cuales tienen una actividad específica. Las actividades pueden ser de coordinación, mapeo o reducción. Por cada capa puede existir uno o más de los siguientes actores: Actor coordinador, Actor MapReduce, Actor de mapeo, Actor de reducción y un Actor que aplica otra fase de reducción llamada agregación. El Actor coordinador es el encargado de mandar el trabajo al Actor MapReduce. El Actor MapReduce es encargado de organizar a los demás actores para aplicar las fases de mapeo y reducción y esperar por la respuesta. El flujo de trabajo es el siguiente:

1. El Actor coordinador recibe la petición e inicia el flujo de trabajo.
2. El Actor coordinador envía una porción de trabajo al Actor MapReduce.
3. El Actor MapReduce manda los datos de entrada al Actor de mapeo.
4. El Actor de mapeo divide los datos recibidos y los delega a componentes de capa inferior. Regresa el conjunto de pares llave - valor.
5. El Actor MapReduce recibe el resultado del Actor de mapeo y lo envía al Actor de reducción.
6. El Actor de reducción divide los datos recibidos y los delega a componentes de capa inferior. Regresa el conjunto de pares llave - valor reducido.
7. El Actor MapReduce recibe el resultado del Actor de reducción y lo envía al Actor de agregación.
8. El Actor de agregación recibe todos los resultados provenientes de los actores de reducción. Consolida los resultados aplicando una función de agregación en un nuevo conjunto de pares llave - valor .
9. El Actor de agregación envía la respuesta al Actor MapReduce.
10. El Actor MapReduce envía la respuesta al Actor coordinador.
11. El Actor coordinador recibe el resultado y devuelve la respuesta a la petición.

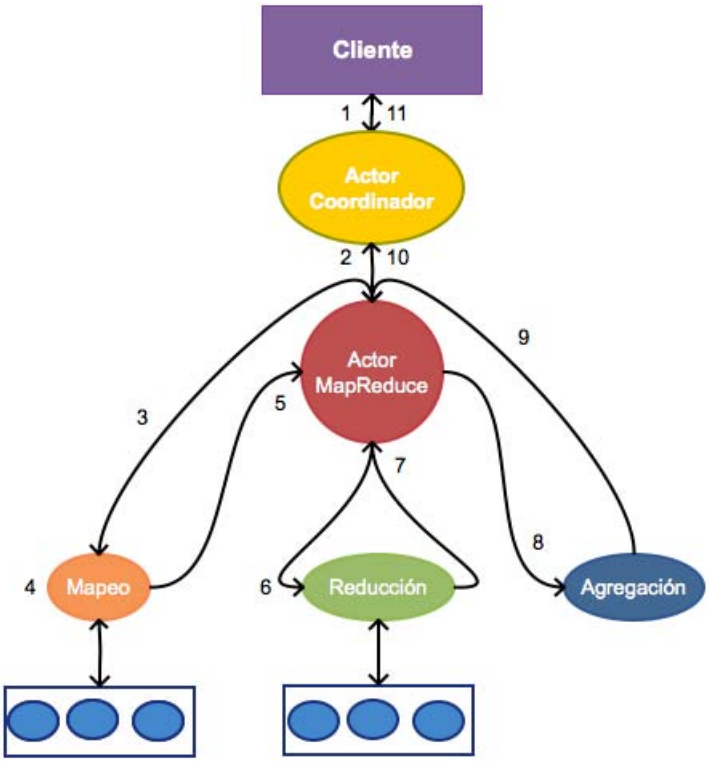

Figura 4.3: Diagrama de bloques de un modelo MapReduce

4.3. Resumen

En este capítulo se describen dos Arqutitecturas de Software para aplicaciones Web que exponen servicios REST. Se proponen dos soluciones. La primer solución se describe mediante el patrón arquitectónico Layers [7] y no utiliza procesamiento en paralelo. Las tareas se ejecutan de manera secuencial. Se identifican algunos retos en la primer solución. También se describe una solución con el mismo objetivo utilizando el patrón arquitectónico Parallel Layers [24]. Se pueden identificar algunas tareas que se pueden ejecutar de manera simultánea lo cual puede utilizar mejor los recursos y mantener el tiempo de respuesta.

Como unidad básica de procesamiento se consideran los actores. Los actores se proponen como componentes de capa que se ejecutan en paralelo.

Capítulo 5

Caso de Estudio: Contador de palabras

Este capítulo tiene como objetivo realizar una evaluación de las Arquitecturas de Software propuestas en el capítulo anterior. Se realizan simulaciones para un caso de estudio de un contador de ocurrencias de cada palabra de un archivo de texto. La funcionalidad se expone como servicio REST en una aplicación Web. En la primer sección se describe un caso de estudio de una aplicación Web que ofrece la funcionalidad de un contador de palabras mediante un servicio REST. En la siguiente sección se describe y justifican algunas decisiones por las cuales se diseña una Arquitectura de Software para aplicaciones Web con procesamiento en paralelo. En la siguiente sección se describe una biblioteca que se utiliza en ambas implementaciones que realiza las funciones de mapeo y reducción. Enseguida se describe la implementación de la Arquitectura de Software con el patrón de Layers [7] que procesa las peticiones de forma secuencial. Posteriormente se describe una Arquitectura de Software con el patrón de Parallel Layers [24] que permite dividir el trabajo a realizar en diferentes tareas, y algunas de las tareas se pueden procesar en paralelo.

Para la implementación se utiliza la plataforma Java. Algunas de las características del lenguaje, bibliotecas y herramientas utilizadas se mencionan en el apéndice A Herramientas para el desarrollo. El código fuente se puede consultar en el apéndice B Código Fuente.

5.1. Características Funcionales de un contador de palabras

Para el caso de estudio se propone una aplicación Web que le permita a un cliente enviar un archivo y obtener como respuesta el número de ocurrencias de cada palabra. Las características del archivo son las siguientes:

- El archivo debe tener el formato de compresión Zip.
- El archivo comprimido debe ser texto plano en formato ASCII.
- El tamaño máximo del archivo es 2 MB.

El cliente debe construir la petición HTTP con el texto a analizar como archivo adjunto. Las características de la petición son las siguientes:

- Debe utilizar el método POST del protocolo HTTP.
- Debe enviar el archivo como adjunto con las especificaciones mencionadas anteriormente.
- El contenido de la petición debe ser especificado como archivo de aplicación Zip.
- El nombre del parámetro del nombre del archivo debe ser "MapReduceText.zip"

El cliente debe conocer la dirección URL del servicio y realizar una petición con los parámetros definidos en la tabla 5.1.

Tabla 5.1: Servicio REST para el conteo de palabras

Descripción	Método HTTP	URL
Servicio Web REST para contar palabras	POST	http://servidor_ip:puerto/paralelo/contadorActores

Un ejemplo de una petición se muestra en el script 5.1. La dirección del servidor es 127.0.0.1 y el puerto es 8080. El nombre del archivo a enviar es el mismo que el parámetro del nombre del archivo de la petición.

Script. 5.1: Ejemplo de una petición HTTP POST con archivo adjunto

```
POST http://127.0.0.1:8080/paralelo/contadorActores
POST data:
--mAcC7VUsa4icoK8QOptP2HmK9yrBLf1
Content-Disposition: form-data; name="MapReduceText.zip"; filename="MapReduceText.
    zip"
Content-Type: application/zip
Content-Transfer-Encoding: binary
<actual file content, not shown here>
--mAcC7VUsa4icoK8QOptP2HmK9yrBLf1--
        \caption{Diagrama de bloques de un ejemplo para contar de palabras.}
```

5.2. Definición de un modelo Map Reduce para contar palabras

El conteo de palabras del texto se realiza mediante un modelo MapReduce. Las divisiones del trabajo para las funciones de mapeo y de reducción se asignan mediante el número de líneas y palabras. Las consideraciones para realizar las divisiones del archivo de texto son las siguientes:

- Una línea se considera como una secuencia de caracteres separados por el carácter de salto de línea.
- Una palabra se considera como una secuencia de caracteres diferentes del carácter de espacio en blanco (espacios, tabulaciones) separados por caracteres de espacio en blanco.
- Las palabras sin significado no son consideradas. Por ejemplo, artículos o preposiciones.

De acuerdo al escenario de aplicación descrito en Capítulo 4, los pasos de un modelo de MapReduce para contar palabras es el siguiente:

1. El cliente realiza una petición Web a la aplicación con las características descritas en la Sección 5.1.

2. El texto a procesar se divide en líneas de palabras. Cierto número de líneas es enviado a a una instancia de la función de contar palabras.

3. Cada instancia de la función de contar palabras divide la porción de texto recibida en líneas y cada una de ellas las envía a una instancia de la función de mapeo.

4. Cada instancia de la función de mapeo recibe una línea y lo divide en palabras. La función genera un conjunto de valores llave - valor intermedio. La llave del conjunto generado es la palabra y valor establecido es 1. El resultado es enviado a una instancia de una función de reducción.

5. En la instancia de una función de reducción los valores con la misma llave se agrupan y se suman. Se genera un nuevo conjunto de valores llave - valor intermedio. La llave es una palabra y el valor es el número de ocurrencias de la palabra en el texto. El resultado de la instancia de la función de reducción es enviado a la función de agregación.

6. La función de agregación recibe el resultado de todas las instancias de la función de reducción y las consolida en un resultado final.

7. Cada instancia de la función de agregación envía el resultado a la función de contar palabras.

8. La función de contar palabras recibe todos los resultados de la fase de agregación y los consolida.

9. La función de contar palabras regresa el resultado final.

En la figura 5.1 se muestra un diagrama de bloques de un ejemplo del funcionamiento del modelo MapReduce para contar palabras. A continuación se describe el funcionamiento del modelo MapReduce para contar palabras en una aplicación Web.

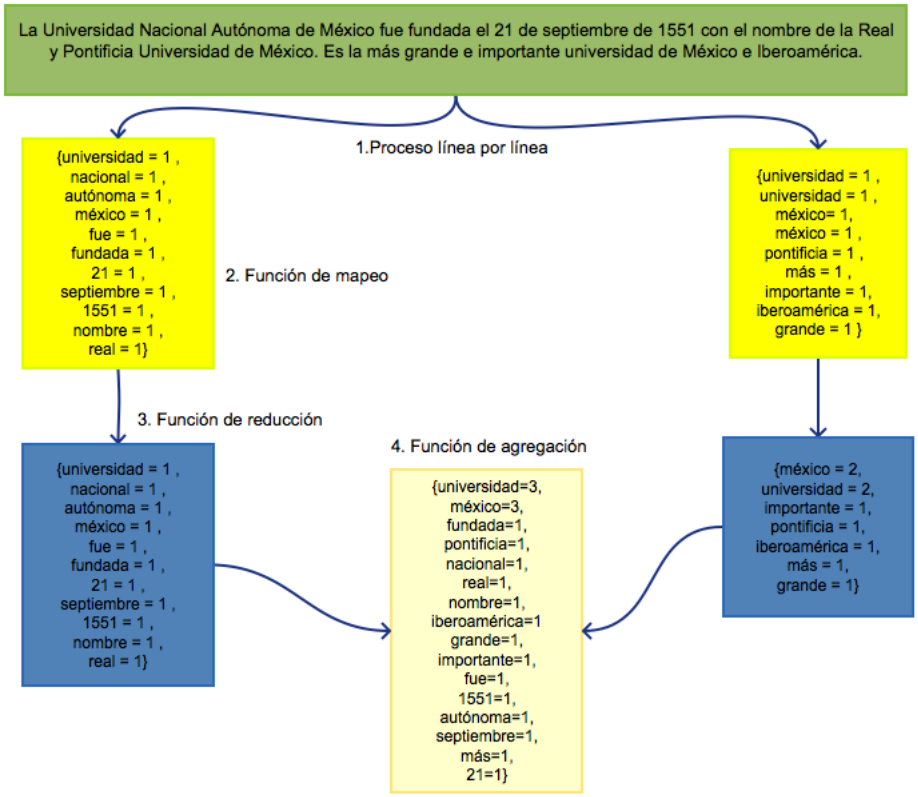

Figura 5.1: Diagrama de bloques de un ejemplo para contar de palabras.

1. El cliente realiza una petición Web a la aplicación.

2. El componente de interface de servicio Web recibe la petición HTTP. Transforma el texto en la petición recibida. Divide en porciones iguales el texto de acuerdo al número de líneas. Una porción del texto es enviada a una instancia de actor Coordinador.

3. El actor Coordinador procesa línea por línea la porción de texto recibida. Cada línea se envía a una instancia de actor MapReduce.

4. El actor MapReduce envía al actor de mapeo la línea recibida.

5. El actor de mapeo recibe una línea de texto y la delega a componentes de capa inferior. Cada componente de capa inferior construye una estructura llave -

valor. La llave es la palabra y el valor para el mapeo es 1 por cada palabra. Se devuelven los resultados al actor MapReduce.

6. El actor MapReduce recibe el resultado del actor de mapeo y lo envía al actor de reducción.

7. El actor de reducción recibe una estructura de entrada y la delega a componentes de capa inferior. Cada actor realiza el conteo de palabras iguales dentro de la estructura generada por el mapeo. Genera una estructura llave - valor. La llave es la palabra y el valor es el número de veces que aparece en la estructura de entrada. La estructura de resultado representa el número de veces que aparece cada palabra en una sola línea. Regresa el conjunto de pares llave - valor reducido al actor MapReduce.

8. El actor MapReduce recibe los resultados del actor de reducción y lo envía al actor de agregación.

9. El actor de agregación realiza la consolidación de todos las estructuras recibidas. Se cuentan el número de veces que aparece cada palabra en cada resultado del proceso de reducción. Reúne los resultados de los actores de reducción. Genera un conjunto de datos llave - valor con el resultado final y lo envía al actor MapReduce.

10. El actor MapReduce recibe el resultado del actor de agregación y lo envía al actor Coordinador.

11. El actor Coordinador consolida los resultados de las instancias de actor MapReduce y devuelve el resultado al componente de interface.

12. El componente de interface regresa el resultado final de la petición al cliente.

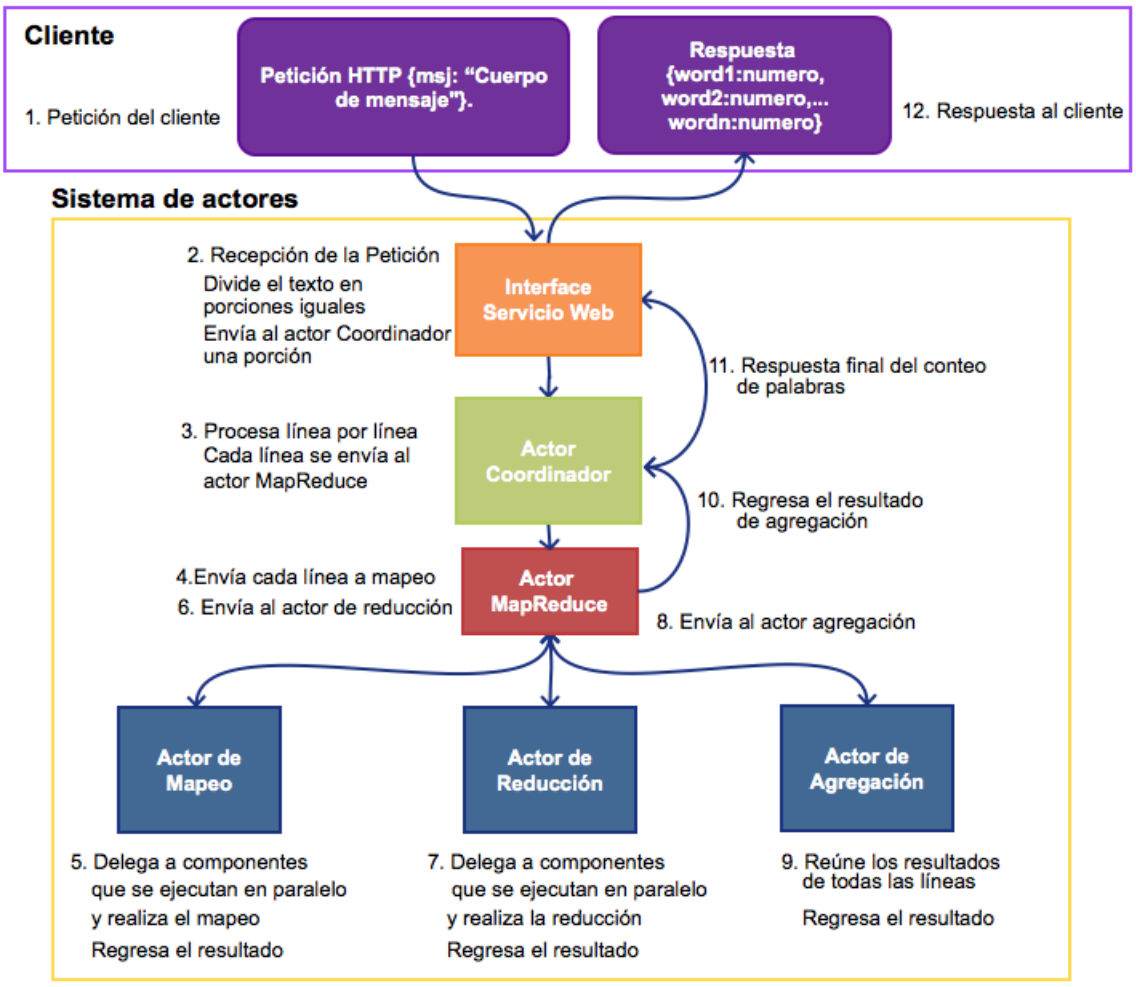

Figura 5.2: Diagrama de bloques de una petición para contar de palabras.

5.3. Justificación en las decisiones de diseño

A continuación se mencionan algunas justificaciones de la arquitectura propuesta:

- El actor interface es el encargado de exponer los servicios REST y aceptar las peticiones HTTP de los clientes. Debe transformar la petición para que se pueda manejar dentro del sistema.
- La comunicación asíncrona entre los actores evita bloqueos. Se aprovechan mejor los recursos otorgados. Cuando ocurre un error o fallo no afecta a los demás componentes del sistema.

- La comunicación por medio de mensajes evita un acoplamiento fuerte entre las capas del sistema. Cada componente de capa envía su mensaje y continúa con su trabajo. Solo debe conocer la dirección del actor destino. El actor destino puede ejecutarse en el mismo servidor o en otro diferente.

- Se fomenta la escabilidad mediante la implementación de actores. Se pueden crear de manera dinámica o estática.

- Se pueden realizar tareas en paralelo dividiendo y delegando actividades a actores de componentes de capa inferiores. De este manera se puede reducir el tiempo de respuesta.

5.4. Estructura General

En la figura 5.3 se presenta un diagrama de paquetes de los tres componentes principales de la implementación. Cada uno de los componentes se describe a continuación.

Paquete utilidad. Los componentes de este paquete son utilizados por ambas Arquitecturas de Software propuestas. Contiene clases con las funciones de mapeo, reducción y agregación para realizar el conteo de palabras. También contiene clases que representan mensajes que se intercambian entre las diferentes capas.

Paquete secuencial. Este paquete contiene los componentes de la implementación utilizando el patrón arquitectónico Layers [7]. Los elementos del paquete de servicios se encargan de exponer el servicio REST, atender y responder las peticiones realizadas. Se delegan tareas a los componentes del paquete de negocios para realizar el conteo de palabras de una petición.

Paquete paralelo. Este paquete contiene los componentes de la implementación utilizando el patrón arquitectónico Parallel Layers [24]. Los elementos del paquete de servicios se encargan de exponer el servicio REST, atender y responder las peticiones realizadas. Se delegan tareas a los componentes del paquete de actores para realizar el conteo de palabras de una petición.

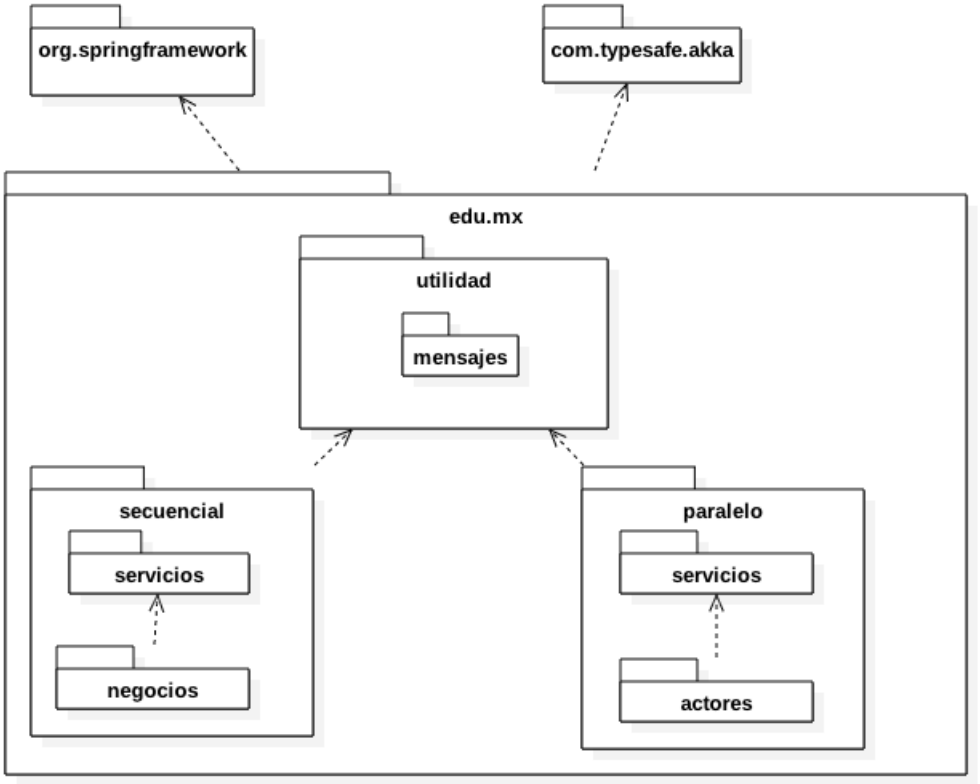

Figura 5.3: Diagrama de paquetes de los componentes de la implementación.

A continuación se describen cada una de las clases de cada paquete. Se omiten algunos métodos de acceso, parámetros de métodos y constructores para facilitar su legibilidad.

5.4.1. Biblioteca de utilidad

En la figura 5.4 se presenta un diagrama de clases del paquete utilidad. Cada una de las clases se describe a continuación:

Palabra. Representa una palabra obtenido de una línea de texto. Contiene la palabra que se está procesado y el número de ocurrencias que existen de acuerdo a la fase de MapReduce.

MapaPalabra. Representa la salida que se obtiene del proceso de mapeo del modelo MapReduce.

MapaReducido. Representa la salida que se obtiene del proceso de reducción del modelo MapReduce.

Resultado. Representa al resultado final del conteo de palabras después de la función de agregación.

UtileriaPalabra. Contiene las funciones de mapeo, reducción y agregación utilizadas para realizar el conteo de palabras.

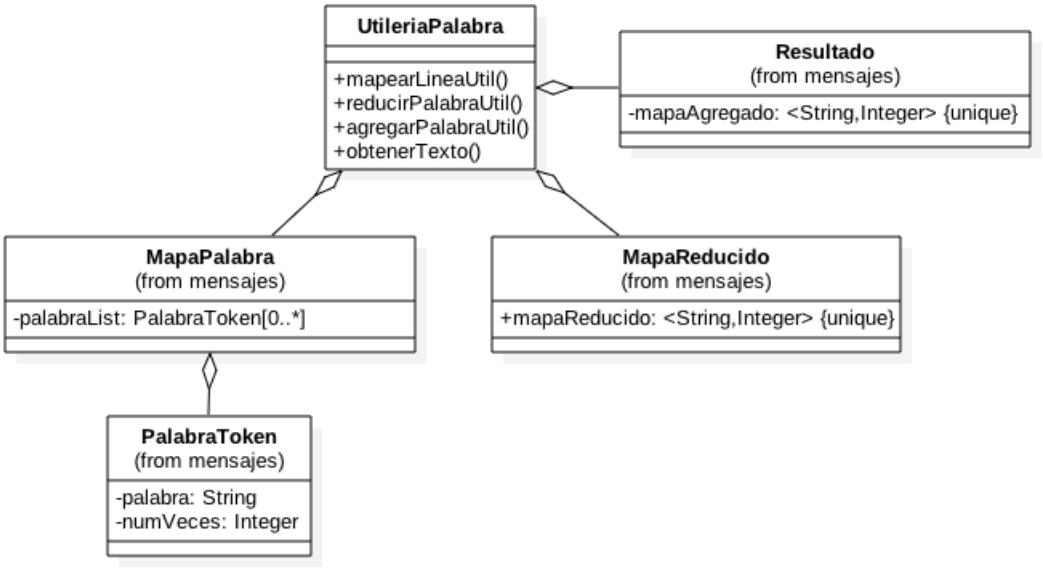

Figura 5.4: Diagrama de clases del paquete de utilidad.

5.4.2. Implementación utilizando el patrón Layers

Las clases de implementación utilizando el patrón Layers [7] se organizan en un paquete llamado secuencial. En la figura 5.5 se presenta un diagrama de clases del paquete secuencial. Cada una de las clases se describe a continuación:

ControladorSecuencialREST. Clase que expone un método como servicio REST para contar palabras. Recibe la petición del cliente y envía la respuesta. Esta clase delega las operaciones del conteo de palabras a la clase MapReduceNegocio.

MapReduceNegocio. Clase que realiza el conteo de palabras de manera secuencial. Delega operaciones a las clases de utilidad. El resultado lo devuelve a la clase ControladorSecuencialREST.

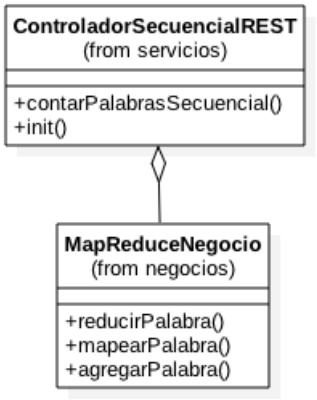

Figura 5.5: Diagrama de clases del paquete secuencial.

5.4.3. Implementación utilizando el patrón Parallel Layers

Las clases de implementación utilizando el patrón Parallel Layers [24] se organizan en un paquete llamado paralelo. En la figura 5.6 se presenta un diagrama de clases del paquete paralelo. Cada una de las clases se describe a continuación:

ControladorActoresREST. Clase que expone un método como servicio REST para contar palabras. Recibe la petición del cliente y envía la respuesta. Divide el texto a procesar en porciones iguales y reparte el trabajo entre las clases a las clases del subpaquete de actores.

CoordinadorActor. Clase que representa al actor que coordina el procesamiento de cada línea de texto de la porción recibida. Crea, delega y coordina las actividades del actor MapReduce.

MapReduceActor. Se encarga de crear un sistema de actores para realizar las tareas del modelo MapReduce línea por línea. Delega y coordina las actividades de los otros actores.

MapearActor. Clase que representa al actor que realiza la función mapeo. El resultado lo devuelve a la clase MapReduceActor.

ReducirActor. Clase que representa al actor que realiza la función reducción. El resultado lo devuelve a la clase MapReduceActor.

AgregarActor. Clase que representa al actor que realiza la función de agregación. El resultado lo devuelve a la clase MapReduceActor.

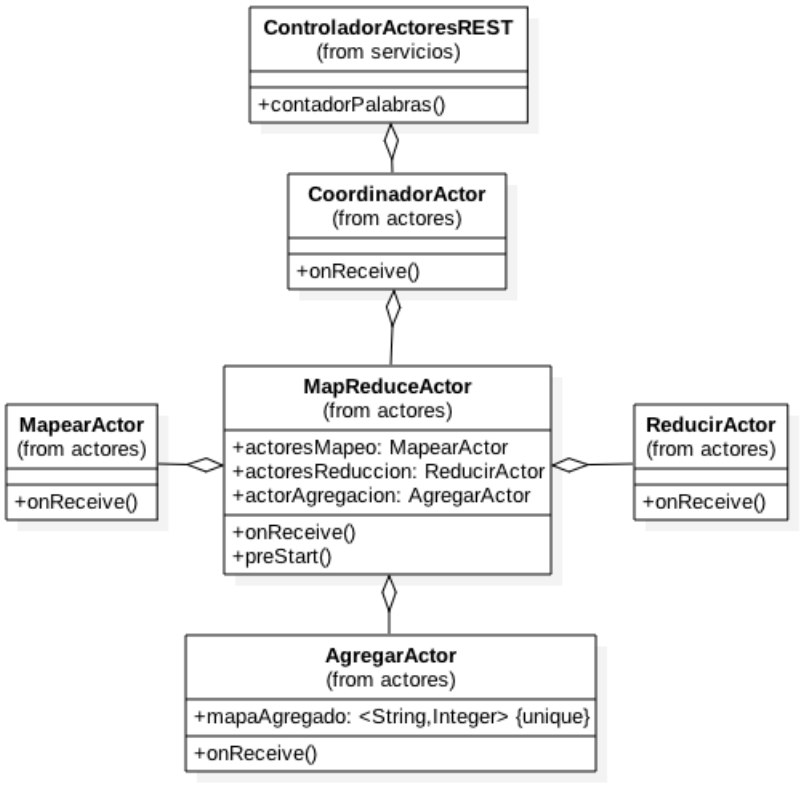

Figura 5.6: Diagrama de clases del paquete paralelo.

5.5. Ejecución de pruebas de la implementación

La configuración de la implementación de la Arquitectura de Software que utiliza el patrón Layers [7] se realiza de acuerdo a lo que se describe en la Sección 4.1.

La configuración de la implementación de la Arquitectura de Software que utiliza el patrón Parallel Layers [24] y que emplea el procesamiento en paralelo se muestra en la figura 5.7. El archivo a analizar se obtiene de la petición del cliente. El archivo se divide por el número de líneas en dos porciones iguales de acuerdo a lo que se describe en la Sección 5.2. Cada porción se envía a una instancia del actor Coordinador. El actor Coordinador procesa línea por línea la porción del texto recibido. Cada línea se envía a una instancia del actor MapReduce, se encarga de crear y coordinar los actores que realizan las funciones de Map - Reduce. El actor de mapeo y el actor de reducción crean a su vez actores para realizar el trabajo asignado. El número de actores se establece de acuerdo a las características del equipo de pruebas, que se describen en la tabla 5.2. Existe un solo actor de agregación que se encarga de consolidar los resultados.

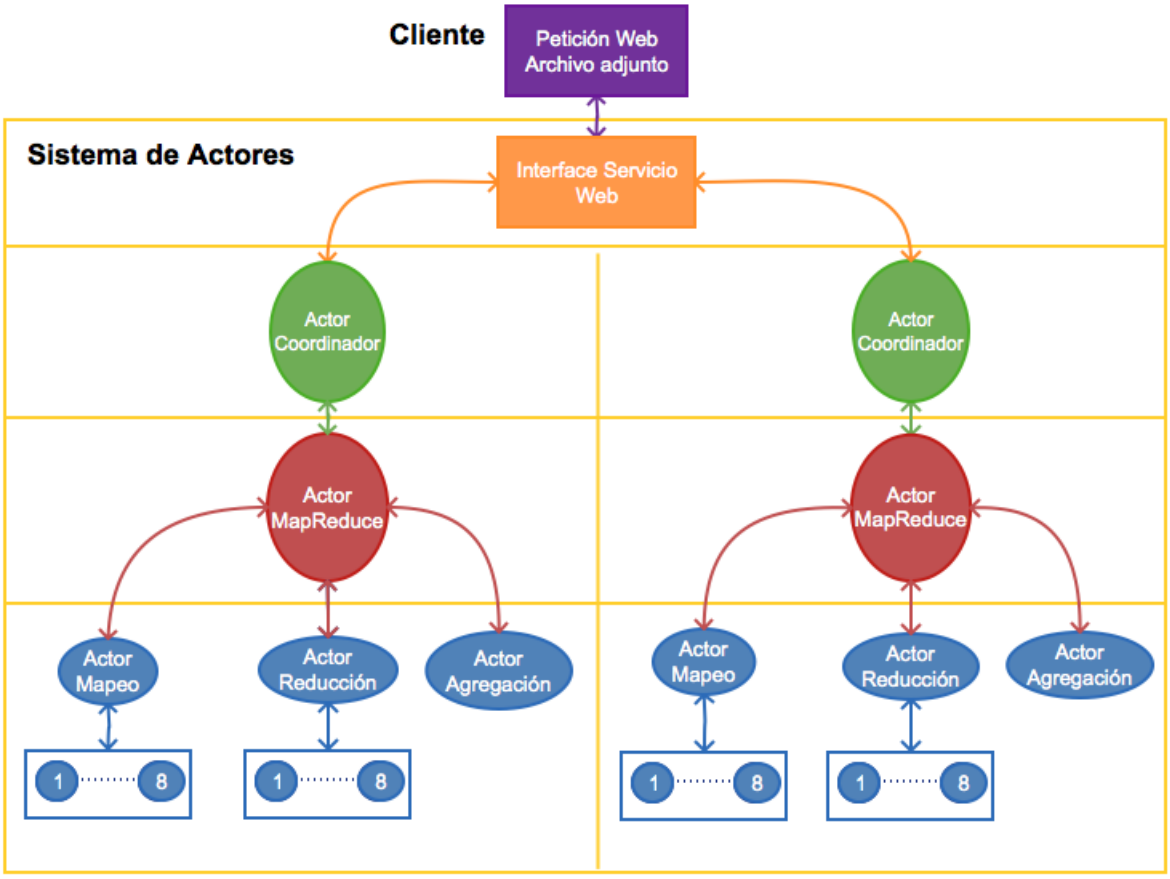

Figura 5.7: Configuración de una implementación con actores.

Para establecer el ambiente de pruebas se realiza un despliegue de las aplicaciones Java en un contenedor de Servlet Tomcat descrito en el apéndice A Herramientas para el desarrollo. A continuación se describen las características de los equipos que se utilizan para realizar las pruebas.

Tabla 5.2: Descripción del Servidor

Recursos de Hardware	
Recurso	**Descripción**
Dirección IP Servidor	132.248.51.117
Memoria RAM del servidor	16 GB
Procesador	8 núcleos 2.0 Ghz
Recursos de Software	
Recurso	**Descripción**
Sistema Operativo	Ubuntu Server 12. 01
Plataforma de desarrollo	Oracle Java 7
Contenedor de Servlets	Apache Tomcat 7

Tabla 5.3: Descripción del Cliente

Recursos de Hardware	
Recurso	**Descripción**
Dirección IP Servidor	132.248.51.117
Memoria RAM del servidor	16 GB
Procesador	8 núcleos 2.0 Ghz
Recursos de Software	
Recurso	**Descripción**
Sistema Operativo	Ubuntu 12.04.4
Plataforma de desarrollo	Oracle Java 7

Se diseñan diferentes planes de pruebas. Cada plan de prueba tiene el mismo número de clientes y el mismo número de repeticiones. El número de líneas en el archivo a analizar es diferente en cada plan. En la tabla 5.4 se muestra la configuración

de los planes de prueba.

Tabla 5.4: Configuración del plan de pruebas

Componentes	**Descripción**
Número de clientes	1 cliente
Número de repeticiones	30
Plan 1	
Líneas en el archivo	10
Plan 2	
Líneas en el archivo	100
Plan 3	
Líneas en el archivo	1000
Plan 4	
Líneas en el archivo	10000

5.6. Resumen

En este capítulo se describen las características principales de la implementación de las Arquitecturas de Software para una aplicación Web utilizando los diseños propuestos en el Capítulo 4. La primer implementación es con el diseño que utiliza el patrón Layers [7]. La segunda es con el diseño que utiliza el patrón Parallel Layers [24] y que incorpora procesamiento en paralelo para atender peticiones. Se propone un modelo de programación MapReduce como caso de estudio y analizar su funcionamiento. El modelo MapReduce propone una solución para contar el número de ocurrencias de cada palabra en un archivo. La funcionalidad es expuesta mediante un servicio Web con arquitectura REST. Se proponen escenarios de prueba para ambas implementaciones y comparar los resultados obtenidos.

Capítulo 6

Resultados experimentales

Este capítulo presenta los resultados de la experimentación en diferentes escenarios aplicados a las arquitecturas que se describen en el Capítulo 5. Se describe una forma para medir el tiempo de respuesta de una aplicación que expone un servicio Web. También se proponen diferentes métricas para evaluar el desempeño de las aplicaciones Web con respecto al tiempo de respuesta de una petición. Finalmente se presenta una comparación con las mediciones obtenidas de ambas aplicaciones.

6.1. Medición de las pruebas

En la figura 6.1 se muestra un diagrama de secuencia que representa una petición desde un cliente a una aplicación por capas que expone un servicio Web. Se considera una sola entrada y existen dos tipos de respuestas. La respuesta donde el procesamiento se realiza con éxito y la respuesta donde ocurren errores.

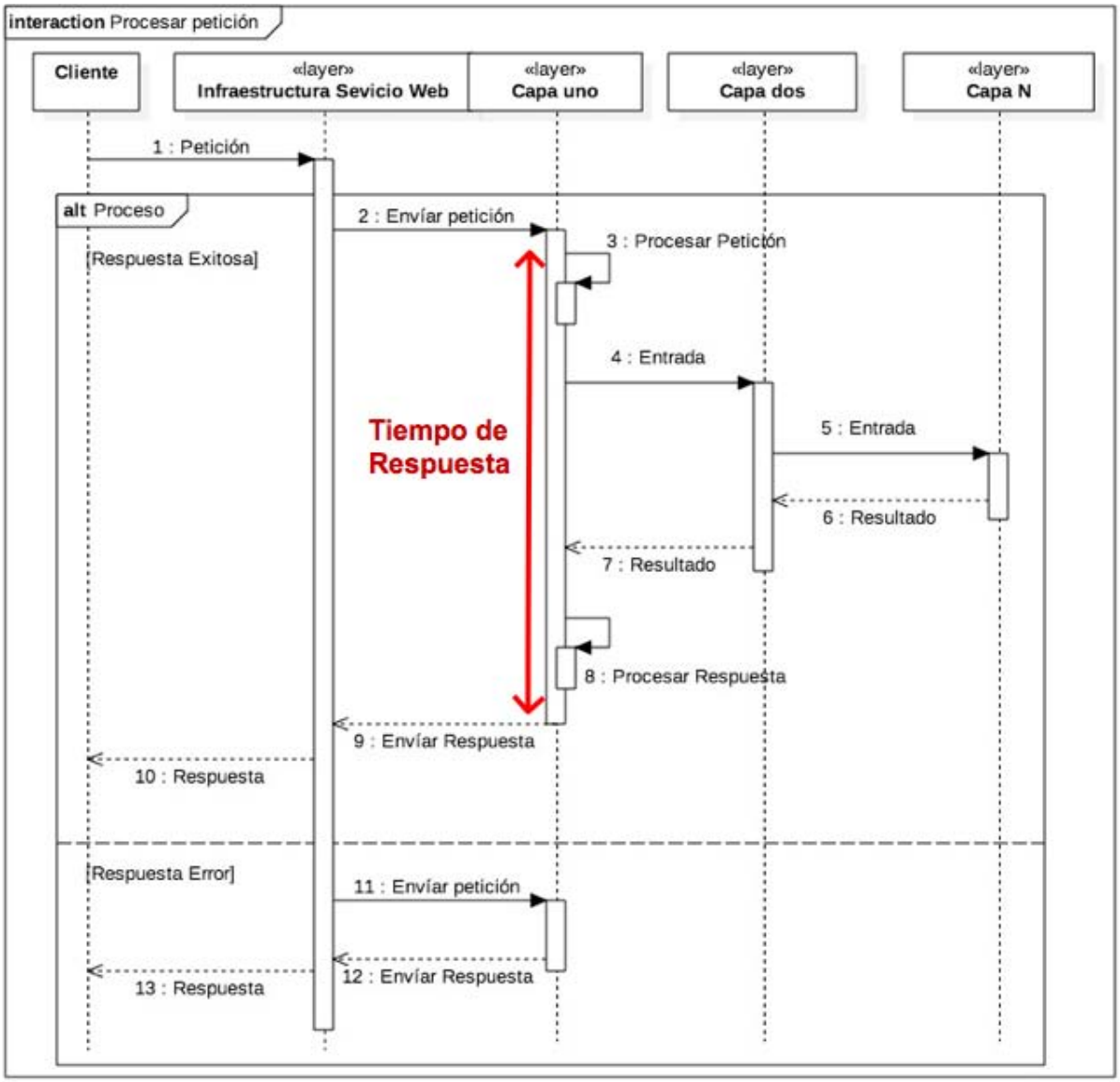

Figura 6.1: Diagrama de secuencia de una petición a un servicio Web.

Para la medición del tiempo de respuesta solo se consideran las respuestas exitosas. Las salidas de error no son consideradas. El tiempo de respuesta es un intervalo de tiempo de una ventana de respuesta dada por un tiempo de inicio y un tiempo de finalización [22].

El momento de inicio de la ventana de tiempo (tiempo inicial) es cuando la petición proveniente del cliente se recibe en la capa de interface Web, y el momento de finalización (tiempo final) es cuando la respuesta proveniente del servicio Web se envía al cliente. El intervalo del tiempo de respuesta es el tiempo entre el momento de finalización y el momento de inicio.

Las consideraciones para medir el tiempo de respuesta se determinan a partir de que la petición se recibe en la capa de interface Web del sistema y el momento en que la información de respuesta esta disponible para su uso inmediato [9]. Bajo este enfoque cualquier detalle de conectividad del cliente no es considerado. El tiempo que puede ser agregado por dispositivos de interconexión quedan fuera del alcance de este trabajo de tesis.

6.2. Métricas de desempeño

A continuación se mencionan las métricas de desempeño que se utilizan para su análisis en el contexto del servicio Web.

Tiempo promedio de respuesta: Se realizan un conjunto de peticiones de manera secuencial hacia el servicios Web, se mide el tiempo de respuesta de cada una y se obtiene el promedio. El número de muestras de peticiones es 30. El número de muestras se establece a 30 para utilizar la distribución normal en los intervalos de confianza. El intervalo de confianza se puede definir como el rango en el cual puede estar el tiempo de respuesta de una petición. Para este trabajo de tesis se establece que el nivel de confianza es de 95 %.

Speedup: Se puede interpretar como la mejora en la velocidad del procesamiento de una petición entre aplicaciones Web que realizan el mismo trabajo bajo las mismas condiciones.

6.3. Resultados experimentales

La tabla 6.1 presenta una comparación de los tiempos de respuesta obtenidos de la ejecución de las pruebas de acuerdo a la descripción de la Sección 5.5.

Tabla 6.1: Tabla comparativa del caso de estudio.

Número de líneas en el archivo	**Tiempo promedio implementación utilizando el patrón Layers en segundos**	**Tiempo promedio implementación utilizando el patrón Parallel Layers en segundos**
10	1.3642 ± 0.0040	0.7779 ± 0.0030
100	14.9095 ± 0.0280	7.0329 ± 0.0140
1000	138.2425 ± 0.06500	64.2291 ± 0.0280
10000	1381.2317 ± 5.7030	638.0194 ± 1.3363

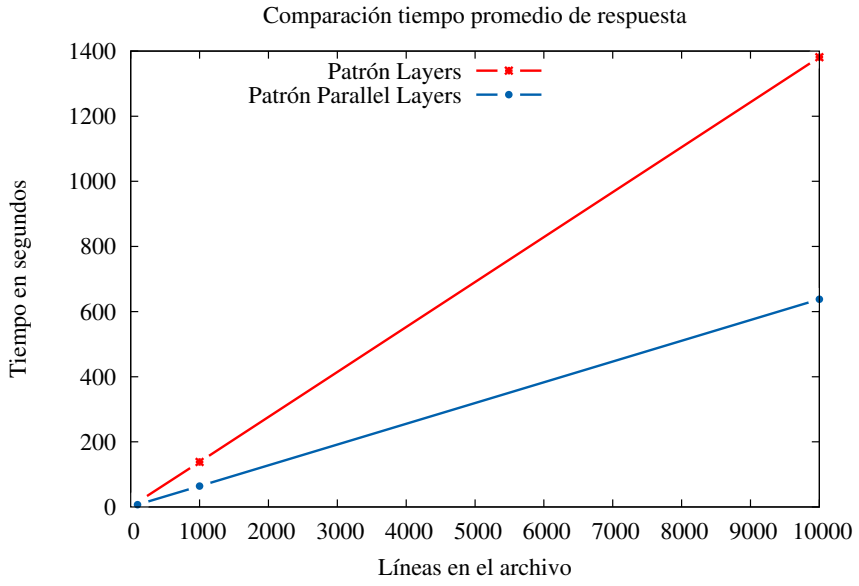

Figura 6.2: Comparación de los tiempos promedio de ejecución.

En la figura 6.2 se muestran dos líneas casi rectas. El tiempo de ejecución total va aumentando conforme la carga de trabajo aumenta y la comunicación entre componentes es constante sin afectar al tiempo de procesamiento.

En la tabla 6.2 se muestra el aumento de velocidad en el tiempo de respuesta que se obtiene del procesamiento utilizando la implementación con el patrón Parallel Layers [24] con respecto a la implementación utilizando el patrón Layers [7] considerando los resultados de la tabla 6.1.

Tabla 6.2: Tabla comparativa Speedup.

Número de líneas en el archivo	**Speedup**
10	1.75
100	2.11
1000	2.15
10000	2.16

6.3.1. Relación entre la división de la carga de trabajo y el desempeño

En la sección anterior se realiza una comparación entre las diferentes formas de procesamiento de una petición a una aplicación Web. En la configuración de la aplicación que se describe en la Sección 5.5 se divide la carga de trabajo en dos partes iguales para realizar el procesamiento. Esta división es de acuerdo a las características y necesidades del problema. La Arquitectura de Software que utiliza el patrón Parallel Layers [24] puede modificarse de acuerdo a las necesidades de paralelización.

Para analizar el impacto de la división de la carga del trabajo en las métricas de desempeño se propone un nuevo experimento. El experimento consiste en realizar una división de la carga de trabajo en 4 partes iguales. La configuración de esta implementación se describe en la figura 6.3. A esta nueva configuración de la aplicación con procesamiento en paralelo se le pueden aplicar los mismos escenarios de prueba que se describen en el Capítulo 5 y aplicar el mismo método de medición propuesto en la Sección 6.1.

El objetivo del experimento es conocer cómo afecta al desempeño la división del trabajo. Se realiza la comparación entre el procesamiento de la implementación que utiliza el patrón Layers [7], el procesamiento de la implementación con el patrón Parallel Layers [24] con 2 y 4 divisiones de trabajo.

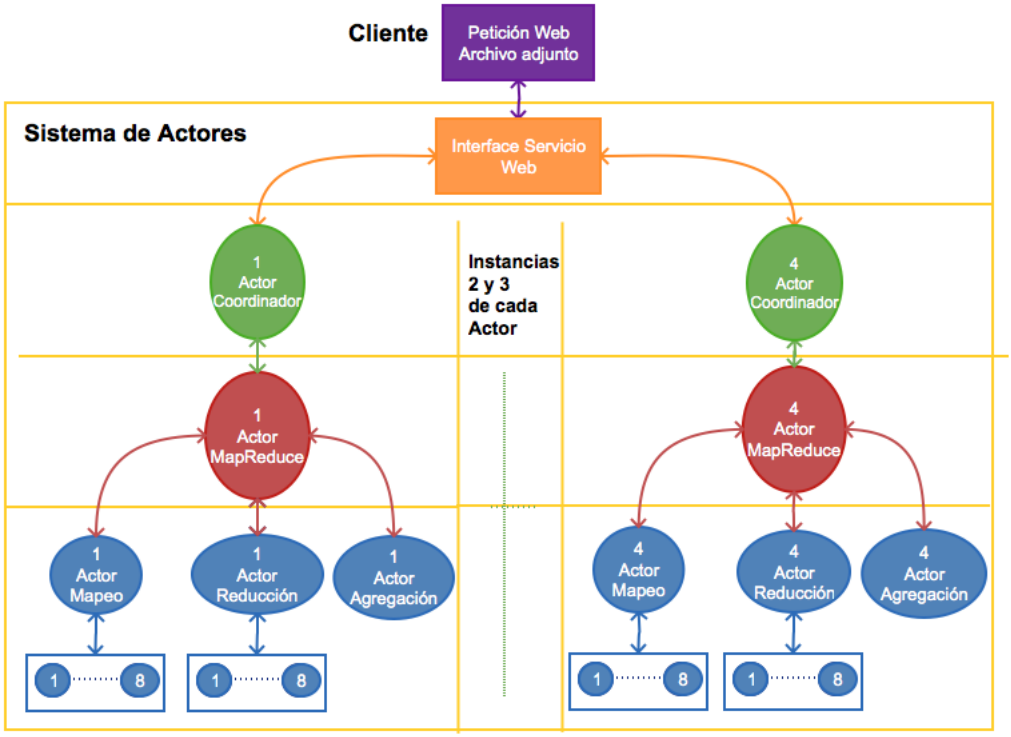

Figura 6.3: Configuración de la aplicación con 4 divisiones.

En la tabla 6.3 se muestra una tabla comparativa de resultados entre las implementaciones utilizando el patrón Parallel Layers [24] que divide en 2 partes y la que divide en 4 partes iguales la carga de trabajo.

Tabla 6.3: Tabla comparativa del caso de estudio del procesamiento en paralelo.

Número de líneas en el archivo	**Tiempo promedio en segundos utilizando el patrón Parallel Layers con 2 divisiones**	**Tiempo promedio en segundos utilizando el patrón Parallel Layers con 4 divisiones**
10	0.7779 ± 0.0030	0.6864 ± 0.0740
100	7.0329 ± 0.0140	3.99 ± 0.0200
1000	64.2291 ± 0.0280	35.5058 ± 0.390
10000	638.0194 ± 1.3363	358.9588 ± 0.867

En la tabla 6.4 se muestra el aumento de velocidad en el tiempo de respuesta que se obtiene del procesamiento en paralelo con 4 divisiones con respecto al procesamiento en paralelo con 2 divisiones considerando los resultados de la tabla 6.3.

Tabla 6.4: Tabla comparativa Speedup de 2 divisiones y 4 divisiones.

Número de líneas en el archivo	**Speedup**
10	1.13
100	1.76
1000	1.80
10000	1.77

En la figura 6.4 se muestra una gráfica considerando los tiempos de respuesta de cada una de las implementaciones.

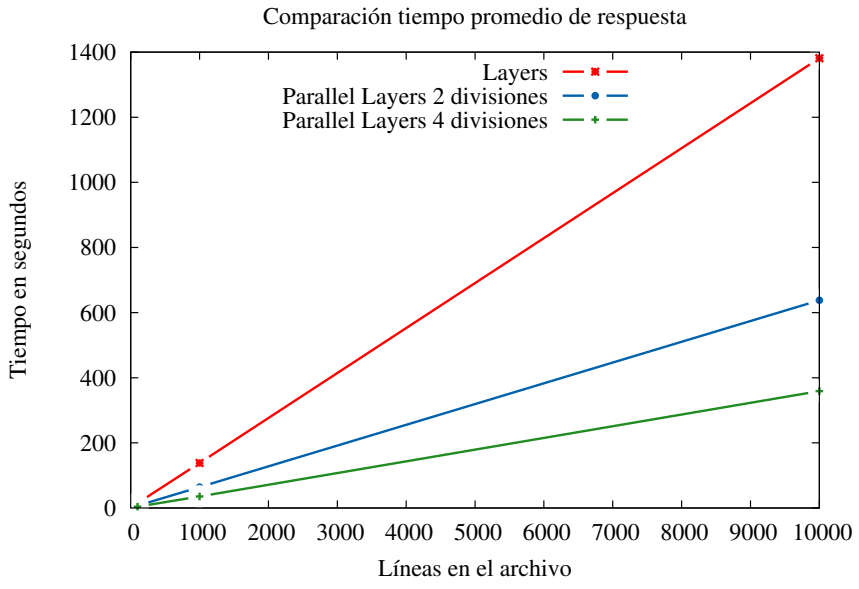

Figura 6.4: Comparación de los tiempos promedio de ejecución.

6.4. Resumen

Este capítulo presenta los resultados experimentales de los escenarios de prueba que se presentan en el Capítulo 5. Se presentan el tiempo promedio de respuesta, un intervalo de confianza y el speedup como métricas para evaluar el desempeño de una aplicación Web. También se presenta una comparación de las medidas obtenidas de cada escenario de prueba.

Capítulo 7

Conclusiones

En este capítulo se realiza una evaluación de la Arquitectura de Software que se describe en el Capítulo 4 del presente trabajo de tesis. Se presenta una recapitulación de la hipótesis y de las contribuciones con el objetivo de realizar una análisis crítico en base a los resultados obtenidos de la experimentación. Se realiza una comparación de ambas Arquitecturas de Software propuestas en base a los resultados de la experimentación con la finalidad de mencionar las aportaciones y conclusiones obtenidas de este trabajo de tesis. Finalmente se proponen nuevos objetivos y posibles vías de investigación futura.

7.1. Evaluación de la hipótesis

Se retoma la hipótesis de la tesis que se presenta en el Capítulo 1 :

"¿Es posible proponer una Arquitectura de Software para una aplicación Web, mediante patrones, que permita dividir el trabajo que se realiza en la atención de peticiones, en tareas más simples que se pueden ejecutar de manera simultánea, con el objetivo de mejorar su desempeño?"

7.1.1. Discusión

Uno de los objetivos de la Arquitectura de Software propuesta para una aplicación Web es mejorar su desempeño. Utilizando el patrón arquitectónico Parallel Layers [24] se distribuye la carga de trabajo procesando tareas en forma paralela

lo cual puede provocar que la petición se atienda en un tiempo menor. Para utilizar el procesamiento en paralelo en una aplicación Web se requieren descripciones de Arquitecturas que permitan diseñar Software enfocado a mantener o mejorar el desempeño.

Esta tesis propone un método experimental basado en analizar el desempeño de una Arquitectura de Software utilizando el patrón de diseño Layers [7] que realiza el procesamiento de las peticiones de manera secuencial y otra que utiliza el patrón de diseño Parallel Layers [24] que permite dividir el trabajo en tareas más simples que se pueden procesar en paralelo. Se establece un método de medición para obtener el tiempo de respuesta de cada Arquitectura con el objetivo de compararlos. La Arquitectura que obtenga el menor tiempo de respuesta es la que tiene mejor desempeño como se plantea en el capítulo 6. Uno de los propósitos es conocer si una Arquitectura de Software que permite el procesamiento en paralelo en algunas tareas mejora el desempeño de una aplicación Web.

Para el caso de estudio se establece una solución mediante un algoritmo Map Reduce. Se propone este problema ya que cumple con las propiedades esenciales para satisfacer la hipótesis: (1) Es de fácil comprensión y (2) Se presenta de manera recurrente como solución a un procesamiento en paralelo de información.

No existe relación entre el algoritmo de Map Reduce y la Arquitectura de Software propuesta. Por lo tanto, la Arquitectura puede extenderse al diseño de Software donde se requiera la ejecución simultánea de tareas. A continuación se proponen dos ejemplos de aplicación de la Arquitectura de Software descrita en el capítulo 4:

Ejemplo 1. Sistema de mensajería. El ejemplo propone un sistema de mensajería que procesa el envío y recepción diferentes tipos de mensajes donde cada uno cuenta con diferente tiempo de procesamiento. Los tiempos de procesamiento propuestos para cada tipo de mensaje del ejemplo son 2, 5 y 10 segundos respectivamente. Un cliente de origen A quiere mandar un mensaje de cada tipo a un destino B. Si los mensajes se procesan de manera secuencial, el tiempo total es la suma de los tiempos de cada mensaje. Si se procesan de manera paralela, el tiempo total es igual del mensaje con mas tiempo de procesamiento. En este escenario se puede crear un actor para que administre todos los mensajes que vayan dirigidos a un cliente. Se puede crear un actor por

cada tipo de mensajes y realizar el procesamiento en paralelo. La comunicación entre los actores es asíncrona para evitar bloqueos.

Ejemplo 2. Administración de una entidad de negocio. Se propone un sistema que realiza la administración de la información de una entidad de negocio, por ejemplo, la administración de inscripciones de estudiantes a asignaturas en un sistema escolar. Un estudiante solicita una inscripción a una asignatura. Pueden existir cualquier número de asignaturas y cualquier número de estudiantes. Solo el número de estudiantes por cada materia es restringido. Se propone crear un actor administrador por cada nueva asignatura. El actor administrador tiene como estado interno el nombre de la asignatura y el número total de lugares disponibles. El actor administrador puede crear un conjunto de n actores auxiliares de inscripción, los cuales se les asigna una porción de los lugares disponibles y lo almacenan como estado interno. Por cada solicitud de inscripción, el actor administrador la delega a un actor auxiliar, el cual se encarga de realizar la inscripción. De esta manera, se puede tener n número de inscripciones al mismo tiempo. La lógica de concurrencia sobre el número de lugares se simplifica, al no compartir estado, los actores auxiliares no pueden inscribir a estudiantes en lugares que no le fueron asignados y solo pueden procesar una sola inscripción a la vez. Una vez agotados los lugares disponibles, el actor auxiliar notifica al actor administrador mediante un mensaje para ya no recibir mas mensajes de inscripción.

La conclusión obtenida del presente trabajo de tesis es que el desempeño de una aplicación Web se mejora cuando su diseño de Software utiliza un patrón de diseño que permite dividir el trabajo en tareas más simples y algunas de las tareas se pueden procesar en paralelo en la atención de peticiones. Utilizando la paralelización de tareas es posible distribuir la carga de trabajo y disminuir el tiempo de respuesta, lo que justifica y confirma la hipótesis.

Los resultados pueden apoyar al diseño de aplicaciones Web que necesiten dividir el trabajo y utilicen el procesamiento en paralelo para atender peticiones y que tengan por objetivo considerar el tiempo de respuesta como un atributo esencial en el comportamiento de la aplicación.

7.1.2. Análisis e interpretación de los resultados

La evaluación de los resultados que se presenta en el Capítulo 6 muestran que, en todos los casos, el diseño de la aplicación que utiliza al patrón de diseño Parallel Layers [24] tiene un mejor desempeño que la aplicación con el diseño que utiliza el patrón de diseño Layers [7]. Una de las razones principales es que el patrón Parallel Layers [24] divide el trabajo en unidades independientes con tareas especificas que se pueden ejecutar en paralelo, para posteriormente consolidar los resultados y devolver la respuesta al cliente.

En todos los escenarios el tiempo de respuesta de la implementación del diseño que utiliza el patrón Parallel Layers [24] fue menor que el de la implementación del diseño que utiliza el patrón Layers [7] . De igual manera, el rango de los intervalos de confianza es menor. Es decir, la aplicación con el procesamiento en paralelo atiende mas rápido las peticiones y la variación con respecto al tiempo promedio de respuesta es menor.

También se demuestra que la división del trabajo en más porciones mejora al desempeño. Se realizan experimentos donde el procesamiento en paralelo que divide el trabajo 4 partes iguales tiene un menor tiempo de respuesta que el procesamiento que lo divide en 2 partes iguales, aunque ello implica mayor utilización de recursos.

El speedup del procesamiento en paralelo con respecto al secuencial fue mayor en todos los escenarios. La aplicación en paralelo atiende las peticiones alrededor del 200 % mas rápido que el procesamiento secuencial. Este comportamiento es similar cuando se comparan el procesamiento en paralelo con 2 y 4 divisiones de trabajo respectivamente. El procesamiento en paralelo con 4 divisiones se ejecuta alrededor del 150 % mas rápido que el procesamiento en paralelo con 2 divisiones.

7.2. Consideraciones

A continuación se describen las consideraciones que reflejan las ventajas y desventajas de la Arquitectura de Software propuesta:

7.2.1. Ventajas

El tiempo de respuesta a una petición se reduce. Esto se debe a que la Arquitectura de Software promueve la ejecución de tareas en paralelo. El trabajo es dividido en diferentes subtareas de menor complejidad y estas son asignadas a diferentes componentes de capa que tienen la misma asignación que se ejecutan de manera simultánea. Una vez que cada unidad termina su trabajo, devuelven sus resultados parciales. Los resultados parciales son consolidados para formar una solución final que se devuelve al cliente como respuesta.

El throughput de la aplicación aumenta. Al reducir el tiempo de respuesta de una petición, el número de peticiones que la aplicación puede atender en un intervalo de tiempo se incrementa.

La administración de recursos se optimiza. La Arquitectura de Software puede implementarse de distintas manera y conforme a las necesidades del diseñador, por ejemplo, asignar de una mayor cantidad de unidades de trabajo a una tarea que lo necesite.

7.2.2. Desventajas

El diseño de una aplicación puede ser más exhaustivo. La partición de una tarea en varias subtareas de menor complejidad es un reto a considerar. No todas las tareas se pueden dividir y este análisis puede requerir mucho tiempo. También se añade complejidad en la comunicación y coordinación entre capas. La comunicación se realiza mediante mensajes. Esto provoca la generación de mensajes de control adicionales a los mensajes de la realización del trabajo.

La creación dinámica de recursos puede provocar que se agoten. Los componentes de capa pueden crearse bajo demanda. Cada componente de capa utiliza recursos. Si existe una tarea con una complejidad considerable, se le pueden asignar más recursos. Pero la asignación de recursos a una sola petición puede ocasionar que se agoten provocando que otras peticiones no sean atendidas.

Mecanismos de detección y recuperación de errores. Aunque el diseño por capas promueve que los errores no se propaguen en todo el sistema, no se describe la solución cuando un componente de capa tiene una falla. La falla puede ser informada a la capa superior y el componente de capa superior puede ejecutar tareas para intentar reparar el error, o en caso contrario, poder escalar nuevamente la falla.

7.3. Comparación con el trabajo relacionado

Se han realizado estudios relacionados con el análisis de desempeño en una aplicación Web anteriormente. La mayoría de los trabajos que se revisaron para este trabajo de tesis se basan en una tecnología, framework, herramientas y/o configuraciones de ambientes de ejecución específicas. Este trabajo se distingue de los anteriores en que se realiza una descripción de una Arquitectura de Software para una aplicación Web. La Arquitectura propuesta puede implementarse de diferentes maneras o modificarse en caso de ser requerido.

A continuación se presenta una comparación con el trabajo relacionado con este trabajo de tesis.

- En el trabajo denominado "Approaches to building high performance Web applications: A practical look at availibity, reliability, and performance"[15] proponen un conjunto de aproximaciones para construir aplicaciones de alto desempeño. Las aproximaciones se enfocan en diferentes atributos como la disponibilidad, confiabilidad y el desempeño. La práctica enfocada el desempeño es la utilización de procesamiento asíncrono de tareas. En este trabajo los autores proponen eliminar o reducir las tareas que forman parte del procesamiento de una petición y que no son esenciales para la atención de la misma. Las tareas que registran ciertos eventos en una bitácora son un ejemplo de ellas.

  La tesis se distingue de este trabajo relacionado en que la mayoría de tareas de la lógica esencial para atender una petición se realizan de manera asíncrona con el fin de ejecutarse en paralelo. Los autores hacen una distinción de tareas que no son esenciales para la atención de una petición. Quitando o reduciendo estas tareas tal vez no reduzcan de manera significativa el tiempo de respuesta de

una aplicación Web. De acuerdo al análisis e interpretación de los resultados, la aplicación Web con procesamiento en paralelo se ejecuta alrededor de un 200 % mas rápido que una tradicional.

- En el trabajo denominado "On Actors and the REST"[21] investiga la relación de los servicios Web REST y el modelo computacional Actor. Describe como las restricciones del modelo computacional Actor pueden ser utilizadas para entender los principios del paradigma REST y propone una notación para describir sistemas REST.

  La tesis se distingue de este trabajo relacionado en proponer un patrón arquitectónico de una aplicación que expone servicios REST que utiliza actores para realizar el procesamiento de las peticiones. Se analiza como organizar a los actores en diferentes capas de acuerdo a su función y se construye una implementación. La implementación se describe en Capítulo 5 con el objetivo de analizar su desempeño.

- En el trabajo denominado "An Approach to Evaluate the Performance of Web Application Systems"[18] se propone una aproximación para evaluar el desempeño de una aplicación Web. Para este trabajo relacionado, el tiempo promedio de respuesta, el número total de peticiones HTTP y throughput se consideran como métricas de desempeño de aplicaciones Web.

  La tesis se distingue de este trabajo relacionado en que se consideran el tiempo promedio de respuesta y el speedup como métricas de desempeño. Se analiza el desempeño desde el momento en que se recibe la petición en la aplicación Web hasta el momento en que la respuesta se envía al cliente. Se analiza sólo tiempo que se emplea en resolver la petición y el tiempo que puede ser agregado por dispositivos de interconexión no se toma cuenta. Esto permite analizar el tiempo de respuesta de la implementación de la Arquitectura de Software propuesta independiente de su ambiente, mientras que el trabajo relacionado propone configuraciones de diferentes ambientes que pueden ser considerados como controlados.

7.4. Resumen de las contribuciones

La conclusión que se obtiene para este trabajo de tesis es que usando una Arquitectura de Software que incorpora patrones de diseño que permiten dividir el trabajo en tareas más simples y que se puedan ejecutar en paralelo en la atención de peticiones en una aplicación Web provoca una reducción del tiempo de respuesta, en comparación con otra Arquitectura que no divide el trabajo. Se concluye que se obtiene un mejor desempeño en la atención de una petición debido a que el procesamiento se divide en tareas de menor complejidad que se pueden ejecutar de manera simultánea, para posteriormente consolidar los resultados y enviar al cliente una solución final.

Las contribuciones que se obtienen de este trabajo de tesis son:

- En el Capítulo 4 se describe una Arquitectura de Software para una aplicación Web que utiliza el patrón de diseño Parallel Layers [24] en la atención de peticiones.
- En el mismo Capítulo 4 se propone utilizar el modelo de concurrencia computacional de Actor como unidad de procesamiento principal.
- En el Capítulo 6 se demuestra que dividir el trabajo en tareas más simples y que algunas de ellas se puedan ejecutar en paralelo en la atención de peticiones mejora al desempeño de una aplicación Web.

7.5. Trabajo futuro

El enfoque del presente trabajo de tesis es analizar el desempeño de una Arquitectura de Software con procesamiento en paralelo para una aplicación Web. El estudio del análisis del desempeño se obtiene mediante mediciones de los tiempos de respuesta en diferentes configuraciones de ambientes de prueba. A continuación se proponen temas de interés que pueden tomar como base este trabajo de tesis y que pueden representar posibles temas de estudio:

- **Análisis de desempeño de una Arquitectura de Software en un Sistema Distribuido.** El estudio y evaluación de una Arquitectura de Software

para una aplicación con procesamiento en paralelo se puede extender a un sistema distribuido. Se pueden considerar variaciones de la arquitectura descrita en este trabajo de tesis. Se pueden realizar comparaciones de escenarios donde los componentes de capa se comunican de manera remota a través de una red de computadoras. La intención es analizar el desempeño de la Arquitectura de Software propuesta en este trabajo en un ambiente distribuido.

- **Análisis de desempeño de una Arquitectura de Software basado en otros patrones arquitectónicos.** El estudio y evaluación de una Arquitectura de Software para una aplicación con procesamiento en paralelo puede ser descrita utilizando otros patrones arquitectónicos diferentes a Parallel Layers. Ejemplo de ellos son Parallel Pipes and Filters [24] o Manager-Workers [24]. La intención es conocer si otros patrones arquitectónicos pueden beneficiar al procesamiento de peticiones de una aplicación Web.

- **Patrones de diseño para corrección y detección de errores.** En el caso de que suceda un error en algún componente de capa, se pueden proponer nuevos patrones de diseño o integrar variaciones de los existentes (Multiple Local Call [24]) para detectar y corregir errores en la aplicación. Estos patrones pueden considerar al modelo computacional Actor como unidad principal de trabajo. La intención es analizar el desempeño de la Arquitectura de Software propuesta con y sin mecanismos de control y detección de errores.

Apéndice A

Herramientas para el desarrollo.

A.1. Plataforma Java

La plataforma Java fue diseñada por Sun Microsystemas como un proyecto en la década de los 90's y fue adquirido por la compañía Oracle en 2010.

Un programa Java puede ejecutarse en diferentes tipo de CPU y diferentes combinaciones de sistemas operativos. Los programas de tecnología Java se compilan utilizando un compilador Java. El formato resultante de un programa Java compilado es el código de byte independiente de la plataforma (bytecode) . Una vez creado el bytecode, se interpreta o es ejecutado por un intérprete conocido como máquina virtual o VM. Una máquina virtual es un programa específico de una plataforma que entiende el bytecode y puede ejecutarlo.

Una máquina virtual recibe su nombre debido a que es una pieza de software que ejecuta código. Esta tarea por lo general es realizada por el CPU de la máquina o hardware. La Máquina Virtual Java (Java Virtual Machine o JVM) es responsable de interpretar código, cargar las clases, y ejecutar programas Java. Un programa de tecnología Java también necesita un conjunto de bibliotecas estándar para la plataforma. Dichas bibliotecas son código escrito previamente y pueden ser utilizadas para crear aplicaciones robustas. En conjunto, el software de la JVM y las bibliotecas estándar Java se conocen como el entorno de ejecución de Java (JRE). Algunas características del lenguaje Java se mencionan a continuación:

Portable. El código fuente se compila a código intermedio (bytecode) el cuál es

interpretado por la máquina virtual de Java.

Multithreading. Soporte múltiples hilos de ejecución (Threads). Puede ejecutar simultáneamente diferentes bloques de código.

Orientado a objetos. Uno de los principales objetivos de la tecnología Java es la creación de objetos, piezas de código autónomos, que pueden interactuar con otros objetos para resolver un problema.

Distribuido. El lenguaje proporciona soporte para tecnologías distribuidas en red. Remote Method Invocation (RMI), CORBA (CORBA), y Universal Resource Locator (URL) son ejemplos de ello.

Para este trabajo de tesis se utiliza la plataforma y lenguaje Java versión 7. Tiene una gran cantidad de bibliotecas, es de fácil aprendizaje y distribución libre.

A.2. Bibliotecas utilizadas

A continuación se mencionan las bibliotecas utilizadas para la construcción de las aplicaciones descritas en el capitulo 5.

A.2.1. Spring Framework

Spring es un framework de código abierto creado originalmente por Rod Johnson. Fue creado para abordar la complejidad del desarrollo de aplicaciones empresariales en Java. La utilidad de Spring no se limita al desarrollo del lado del servidor. Cualquier aplicación Java puede beneficiarse de Spring en términos de simplicidad, la capacidad de prueba, y bajo acoplamiento.

Para simplificar el desarrollo de aplicaciones Java, Spring emplea cuatro estrategias clave:

- Desarrollo ligero y mínimamente invasivo mediante la utilización de clases simples y que no dependen de un framework en especial.
- Bajo acoplamiento a través de la inyección de dependencias y la orientación de interfaces.

- La programación declarativa a través de aspectos y convenciones comunes.
- Reducción de tareas repetitivas mediante aspectos y plantillas.

En este trabajo de tesis se utiliza Spring Framework para construir una aplicación Web. Se utilizan el módulo principal (Spring Core) y el modulo de Spring Web en su versión 4.1.7

A.2.2. Akka

Akka es un conjunto de herramientas de código abierto y que ayuda la construcción de aplicaciones concurrentes y distribuidas en la JVM. Akka es compatible con varios modelos de programación para la concurrencia. Uno de ellos es el modelo de actores. Algunas características son:

Manejo de Concurrencia. Está basada en mensajes y la comunicación se puede realizar de manera asíncrona. La información que se comparte no se puede modificar.

Interacción entre Actores. Los actores pueden interactuar independientemente del host donde se encuentren ubicados mediante de mecanismos de enrutamiento y configuración.

Jerarquía de actores. Los actores se pueden organizar de manera jerárquica. Permitiendo que existan actores dedicados a la supervisión y división de tareas.

Akka se compone de módulos que se distribuyen como archivos JAR. Se puede utilizar como cualquier otra biblioteca en lenguaje Java. La versión 2.11 de la biblioteca es la que se utiliza en este trabajo de tesis. A continuación se muestra un ejemplo de como crear un actor con la biblioteca Akka en el lenguaje Java

Clase ActorEjemplo.java

```
1.  public class ActorEjemplo extends UntypedActor {
2.  public void onReceive(Object mensaje) throws Exception {
3.          if (mensaje instanceof String) {
4.               String saludo = (String) mensaje;
5.               getSender().tell("Hola " + mensaje);
6.          } else
7.               unhandled(message);
8.       }
```

1. Se extiende de la clase UntypedActor que proporciona Akka para construir un actor con una clase Java.

2. Sobre-escribe el método onReceive. Es el único método público y el único medio para comunicarse con el actor. Recibe los mensajes como clases Java.

3. Se verifica de que tipo es el mensaje. Puede recibir diferentes tipos de mensajes y reaccionar a ellos.

4. El mensaje se puede procesar como cualquier clase Java.

5. El método getSender permite obtener el emisor del mensaje. El método tell permite enviar un mensaje. Ambos métodos los proporciona Akka.

6. En caso de que el mensaje sea desconocido, se puede utilizar el método unhandled para ser enviado en un mensaje de error.

A.3. Herramientas complementarias

A continuación se mencionan las herramientas complementarias que se utilizan en el desarrollo de las aplicaciones Web.

A.3.1. Apache Tomcat

Apache Tomcat es un contenedor de aplicaciones Web construidas en Java que se utiliza como servidor. Concretamente sirve para desplegar aplicaciones desarrolladas con tecnologías Web Java (Servlet y JSP). La versión de Tomcat depende de la compatibilidad con la máquina virtual de Java. Para este trabajo de tesis se utiliza la versión Apache Tomcat version 7.0 que implementa la especificación de Servlet 3.0 y JavaServer Pages 2.2

Apéndice B

Código Fuente.

B.1. Biblioteca común

La biblioteca común es una clase que Java que realiza tareas utilizadas por ambas propuestas de solución. Tiene como funciones generar las estructuras llave - valor en cada fase del proceso del MapReduce. A continuación se presenta un fragmento del código java de la biblioteca:

Clase UtileriaPalabra.java

```
public class UtileriaPalabra {

  public static final Logger log = Logger.getLogger(UtileriaPalabra.class);
  public static final int numeroActores;
  private static final int miliSegundos=10;
  public static final String timeOutActores = "2 seconds";
  public static final int awaitActores=500;
  private static List<String> PALABRAS_SIN_SIGNIFICADO_LIST;

  static{
    numeroActores = Runtime.getRuntime().availableProcessors();
    String[] PALABRAS_SIN_SIGNIFICADO = {
    "a","ante","bajo","cabe","con","contra","de",
    "desde","en","entre","hacia","hasta","para","por","según","sin","so","sobre",
    "tras", "el","la","los","las","un","unos","una","unas","al","del","y","es","e"};

    PALABRAS_SIN_SIGNIFICADO_LIST = Arrays.asList(PALABRAS_SIN_SIGNIFICADO);
  }


  public static MapaPalabra mapearLineaUtil(String linea, MapaPalabra mapaPalabra) {
```

```
    StringTokenizer tokenizer = new StringTokenizer(linea);
    try {
      while (tokenizer.hasMoreTokens()) {
        String palabra = tokenizer.nextToken().toLowerCase();
        if (!PALABRAS_SIN_SIGNIFICADO_LIST.contains(palabra)) {
          mapaPalabra.agregarPalabratoken(palabra, 1);
        }
      }

    } catch (InterruptedException e) {
      e.printStackTrace();
      throw new RuntimeException(e);
    }
    return mapaPalabra;
  }

  public static MapaReducidoPalabra reducirPalabraUtil(MapaPalabra mapaPalabra) {
    MapaReducidoPalabra mapaReducidoPalabra = new MapaReducidoPalabra();
    for (PalabraToken palabraToken : mapaPalabra.getPalabraTokenList()) {
      if (mapaReducidoPalabra.tienePalabra(palabraToken.getPalabra())) {
        mapaReducidoPalabra.incrementarPalabra(palabraToken.getPalabra());
      } else {
        mapaReducidoPalabra.agregarNuevaPalabra(palabraToken.getPalabra(), 1);
      }
    }
    return mapaReducidoPalabra;
  }

  public static void agregarPalabraUtil(MapaReducidoPalabra mapaReducidoPalabra,
    Map<String, Integer> mapaAgregado) {

    for (String palabra : mapaReducidoPalabra.getMapaReducido().keySet()) {
      if (mapaAgregado.containsKey(palabra)) {
        Integer conteoFinal = 0;
        conteoFinal = mapaAgregado.get(palabra);
        conteoFinal= conteoFinal + mapaReducidoPalabra.getMapaReducido().get(palabra
            );
        mapaAgregado.put(palabra, conteoFinal);
      } else {
        mapaAgregado.put(palabra,mapaReducidoPalabra.getMapaReducido().get(palabra))
            ;
      }

    }
  }
```

B.2. Implementación Patrón Layers

Esta propuesta de solución no utiliza actores y todo el procesamiento se realiza de manera secuencial. La arquitectura es descrita en la sección 4.1 de una aplicación Web. Existe una capa de servicio que realiza las tareas de procesamiento de cada una de las fases. Cada método utiliza la biblioteca común para trabajar.

Clase MapReduceNegocio.java

```
public class MapReduceNegocio {

  public MapaReducidoPalabra reducirPalabra(MapaPalabra mapaPalabra) {
    MapaReducidoPalabra mapaReducidoPalabra = new MapaReducidoPalabra();
    for (PalabraToken palabraToken : mapaPalabra.getPalabraTokenList()) {
      if (mapaReducidoPalabra.tienePalabra(palabraToken.getPalabra())) {
        mapaReducidoPalabra.incrementarPalabra(palabraToken.getPalabra());
      } else {
        mapaReducidoPalabra.agregarNuevaPalabra(palabraToken.getPalabra(), 1);
      }

    }
    return mapaReducidoPalabra;
  }

  public MapaPalabra mapearPalabra(String texto) {
    return mapearLineaUtil(texto, new MapaPalabra());
  }

  public void agregarPalabra(Map<String, Integer> mapaAgregadoFinal,
    MapaReducidoPalabra mapaReducidoPalabra) {
    agregarPalabraUtil(mapaReducidoPalabra, mapaAgregadoFinal);

  }

}
```

B.3. Implementación Patrón Parallel Layers

La propuesta de la Arquitectura de Software que utiliza Parallel Layers [24] como patrón arquitectónico y propone al modelo computacional de Actor como componente de capa se implementa utilizando la biblioteca Akka descrita en en apéndice A Subsección A.2.2. Cada actor es modelado como una clase Java que extiende de la clase UntypedActor de Akka. Los actores se comunican mediante mensajes. Cada

mensaje es una clase Java que representa unidades de información que se intercambian entre las fases del modelo MapReduce. El actor no comparte estado. Solo tiene un método llamado onReceive. A continuación se presenta cada una de las clases Java de los mensajes.

Clase PalabraToken.java

```
public final class PalabraToken {
        private String palabra;
        private Integer numVeces;
        public PalabraToken(String palabra,Integer numVeces){
                this.palabra=palabra;
                this.numVeces=numVeces;
        }
        public String getPalabra() {
                return palabra;
        }
        public Integer getNumVeces() {
                return numVeces;
        }
                public String toString(){
                return ""+this.palabra+" => "+this.numVeces;
        }
}
```

Clase MapaPalabra.java

```
public final class MapaPalabra {
private List<PalabraToken> palabraTokenList=null;
public MapaPalabra(){
  this.palabraTokenList = new ArrayList<PalabraToken>();
}
public void agregarPalabratoken(String palabra,Integer numVeces){
  PalabraToken palabraTokenInstance = new PalabraToken(palabra,numVeces);
  this.palabraTokenList.add(palabraTokenInstance);
}

public List<PalabraToken> getPalabraTokenList() {
  return palabraTokenList;
}

public String toString(){
  StringBuilder str = new StringBuilder();
  for(PalabraToken palabraToken: palabraTokenList){
    str.append("[ "+ palabraToken+" ] \n");
  }
  return str.toString();
}
}
```

Clase MapaReducidoPalabra.java

```
public final class MapaReducidoPalabra {
private Map<String,Integer> mapaReducido ;
public MapaReducidoPalabra(){
  this.mapaReducido = new HashMap<>();
}
public Map<String,Integer> getMapaReducido(){
  return this.mapaReducido;
}
public void agregarNuevaPalabra(String palabra,Integer valor){
  this.mapaReducido.put(palabra, valor);
}
public void incrementarPalabra(String palabra){
  Integer valor = this.mapaReducido.get(palabra);
  valor++;
  this.mapaReducido.put(palabra, valor);
}
public Boolean tienePalabra(String palabra){
  return this.mapaReducido.containsKey(palabra);
}
public String toString(){
  StringBuilder str = new StringBuilder();
  for(String palabra: this.mapaReducido.keySet()){
    Integer valor = this.mapaReducido.get(palabra);
    str.append("["+palabra+" => "+ valor+"] \n" );
  }
  return str.toString();
}
}
```

Cada fase del modelo MapReduce es implementada con un actor. Existe un actor llamado Coordinador que se encarga de procesar línea por línea una porción del texto y cada línea la envía al actor MapReduce. El actor MapReduce coordina todas las actividades de los actores en intervienen en el proceso de MapReduce. A continuación se presentan las clases Java de los actores. Los actores hacen uso de los métodos de la biblioteca común.

Clase MapearActor.java

```
public class MapearActor extends UntypedActor {
        @Override
        public void onReceive(Object mensaje) throws Exception {
                  if (mensaje instanceof String) {
             String texto = (String) mensaje;
             getSender().tell(mapearLineaUtil(texto,new MapaPalabra()));
         } else unhandled(mensaje);}
  }
```

Clase ReducirActor.java

```
public class ReducirActor extends UntypedActor{
        @Override
        public void onReceive(Object mensaje) throws Exception {
                if (mensaje instanceof MapaPalabra) {
                        MapaPalabra mapaPalabra = (MapaPalabra) mensaje;
                        getSender().tell(reducirPalabraUtil(mapaPalabra));
                } else
                        unhandled(mensaje);
        }
}
```

Clase CoordinadorActor.java

```
public class CoordinadorActor extends UntypedActor {

private final LoggingAdapter log = Logging.getLogger(getContext().system(), "
    CoordinadorActor");
ActorRef mapReduceActor;
 @Override
    public void preStart() throws Exception {
  mapReduceActor = getContext().actorOf(Props.create(MapReduceActor.class),"
      MapReduceActor");
 }
@Override
public void onReceive(Object mensaje) throws Exception {
   if (mensaje instanceof PorcionTexto) {
     PorcionTexto p = (PorcionTexto)mensaje;
     for(String linea: p.getListaTest()){
       mapReduceActor.tell(linea,null);
     }
       }
   else if (mensaje instanceof Resultado) {
     mapReduceActor.forward(mensaje, getContext());
   }
   else
           unhandled(mensaje);
        }
}
```

Clase AgregarActor.java

```
public class AgregarActor extends UntypedActor {

private Map<String, Integer> mapaAgregadoFinal = new TreeMap<String, Integer>();
private Set<String> recibidos = new HashSet<String>();
private ActorRef resultado;
@Override
public void onReceive(Object mensaje) throws Exception {

  if(mensaje instanceof String){
    recibidos.add((String)mensaje);
  }

  if (mensaje instanceof MapaReducidoPalabra) {
    MapaReducidoPalabra mapaReducidoPalabra = (MapaReducidoPalabra) mensaje;
    agregarPalabraUtil(mapaReducidoPalabra,mapaAgregadoFinal);
  } else if (mensaje instanceof Resultado) {
    int tam = mapaAgregadoFinal.size();
    if(recibidos.size()==tam){
      ((Resultado) mensaje).setMapaAgregadoFinal(mapaAgregadoFinal);
      if (resultado == null){
        getSender().tell(mensaje,getSelf());
      }
      else{
        resultado.tell(mensaje,getSelf());
      }
    }
    else{
      resultado = getSender();
      getSelf().tell(mensaje, resultado);
    }
  } else
    unhandled(mensaje);
}
}
```

Clase MapReduceActor.java

```
public class MapReduceActor extends UntypedActor {

private Router routerMapeo;
private Router routerReducir;
private ActorRef actorAgregacion;
private final LoggingAdapter log = Logging.getLogger(getContext().system(), "
    mapReduceActor");

@Override
public void preStart() throws Exception {
    List<Routee> actoresMapeo = new ArrayList<Routee>();
```

```
    List<Routee> actoresReducir = new ArrayList<Routee>();
    for(int i=0;i < (numeroActores) ;i++){
      ActorRef actor = getContext().actorOf(Props.create(MapearActor.class),"
          mapearActor"+i);
       getContext().watch(actor);
      actoresMapeo.add(new ActorRefRoutee(actor));
    }

    for(int i=0;i < (numeroActores);i++){
    ActorRef actor = getContext().actorOf(Props.create(ReducirActor.class),"
        reducirActor"+i);
       getContext().watch(actor);
          actoresReducir.add(new ActorRefRoutee(actor));
      }
    routerMapeo = new Router(new RoundRobinRoutingLogic(), actoresMapeo);
    routerReducir = new Router(new RoundRobinRoutingLogic(), actoresReducir);
    actorAgregacion = getContext().actorOf(Props.create(AgregarActor.class),"
        ActorAgregacion");
    super.preStart();
}

@Override
public void onReceive(Object mensaje) throws Exception {
if (mensaje instanceof String) {
  routerMapeo.route(mensaje, getSelf());
    } else if (mensaje instanceof MapaPalabra) {
      MapaPalabra p = (MapaPalabra)mensaje;
      for(PalabraToken token : p.getPalabraTokenList()){
      actorAgregacion.tell(token.getPalabra(),null);

      }

      routerReducir.route(mensaje,getSelf());
    } else if (mensaje instanceof MapaReducidoPalabra) {
      actorAgregacion.tell(mensaje,getSelf());
    } else if (mensaje instanceof Resultado) {
      actorAgregacion.forward(mensaje, getContext());
    } else if(mensaje instanceof Integer){
      routerMapeo.route(new Broadcast(akka.actor.PoisonPill.getInstance()), null);
      routerReducir.route(new Broadcast(akka.actor.PoisonPill.getInstance()), null);
      actorAgregacion.tell(akka.actor.PoisonPill.getInstance(), null);

    }
else
        unhandled(mensaje);
}

}
```

Bibliografía

[1] Gul A. Agha. Actors: A model of concurrent computation in distributed systems. *Technical Report 844, MIT Artifical Intelligence Laboratory*, 1985.

[2] Christopher Alexander. *The Timeless Way of Building*. Oxford University Press, 1979.

[3] Mario Barbacci, Mark H. Klein, Thomas A. Longstaff, y Charles B. Weinstock. *Quality Attributes*. Software Engineering Institute, 1995.

[4] Len Bass, Paul Clements, y Rick Kazman. *Software Architecture in Practice*. Addison Wesley, 2003.

[5] Mordechai Ben-Ari. *Principles of Concurrent and Distributed Programming*. Prentice Hall, 1990.

[6] Tim Berners-Lee. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*. Harper Collins, 2000.

[7] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, y Michael Stal. *Pattern-Oriented Software Architecture. A system of Patterns*. John Wiley & Sons, 1996.

[8] Common Software Measurement International Consortium. *The COSMIC Functional Size Measurement Method. Measurement Manual*, 2015 (Cuarta edición). http://www.cosmic-sizing.org.

[9] Common Software Measurement International Consortium. *Guideline on Non-Functional & Project Requirements*, 2015 (Primera edición). http://www.cosmic-sizing.org.

[10] World Wide Web Consortium. *Página oficial W3C, Web Services Architecture.*, Febrero 2004 (citado en Octubre del 2015). http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[11] Instituto de Ingeniería Eléctrica y Electrónica. *Página oficial de ISO/IEC/IEEE 42010, Systems and software engineering —Architecture description*, 2011 (citado en Abril del 2015). http://www.iso-architecture.org/ieee-1471/.

[12] Jeffrey Dean y Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM - 50th anniversary issue: 1958 - 2008*, págs. 107–113, 2008.

[13] Thomas Fielding. *Architectural Styles And The Design Of Network-Based Software Architectures*. Tesis Doctoral, Universidad de California, Irvine, 2000.

[14] Erich Gamma, Richard Helm, Ralph Johnson, y John Vlissides. *Design Patterns. Elements of Reusable Object—Oriented Software*. Addison Wesley, 2004.

[15] Brian Goodman, Maheshwar Inampudi, y James Doran. Approaches to building high performance web applications: A practical look at availibity, reliability, and performance. *Architecture of Reliable Web Applications Software*, págs. 112–144, 2007.

[16] Mark Grand. *Patterns in Java 1: A catalog of reusable patterns illustrated with UML*. John Wiley & Sons, 2002.

[17] Carl Hewitt, Peter Bishop, y Richard Steiger. A universal modular actor formalism for artificial intelligence. *IJCAI'73: Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, págs. 235–245, 1973.

[18] Tahani Hussain. An approach to evaluate the performance of web application systems. *Proceedings of International Conference on Information Integration and Web-based Applications & Services*, págs. 692–697, 2013.

[19] Dr. Heinz M. Kabutz. *Blocking queue. The JavaTM Specialists Newsletter*, 2001 (citado en Mayo del 2015). http://www.javaspecialists.eu/archive/Issue016.html.

[20] James F. Kurose y Keith W. Ross. *Computer Networking. A Top Down Approach*. Pearson, 2013.

[21] Janne Kuuskeri y Tuomas Turto. On actors and the rest. *10th International Conference, ICWE 2010*, págs. 144–157, 2010.

[22] Henry H. Liu. *Software Performance and Scalability. A Quantitative Approach*. John Wiley & Sons, 2009.

[23] Sergio Luján Mora. *Programación de aplicaciones web: historia, principios básicos y clientes web*. Editorial Club Universitario, 2002.

[24] Jorge Luis Ortega-Arjona. *Patterns for Parallel Software Design*. John Wiley & Sons, 2010.

[25] Dewayne E. Perry y Alexander L. Wolf. Foundations for the study of software architecture. *Software Engineering Notes vol. 17 no. 4*, págs. 40–52, 1992.

[26] Roger S. Pressman. *Ingeniería del Software. Un enfoque práctico.* Mc Graw Hill, 2010.

[27] Kai Qian. *Software Architecture and Design Illuminated*. Jones and Bartlett Illuminated Series, 2010.

[28] Leonard Richardson y Sam Ruby. *RESTful Web Services*. O?Reilly, 2008.

[29] Gustavo Rossi, Oscar Pastor, Daniel Schwabe, y Luis Olsina. *Web Engineering: Modelling and Implementing Web Applications*. Springer, 2008.

[30] M. Shams, D. Krishnamurthy, y B. Far. A model-based approach for testing the performance of web applications. *Proceedings of the 3rd international workshop on Software quality assurance*, págs. 54–61, 2006.

[31] Mary Shaw y David Garlan. *Software Architecture. Perspective on an emerging discipline.* Prentice Hall, 1996.

[32] Yingxu Wang. *Software Engineeering Foundations. A Software Science Perspective*. Auerbach Publications, 2008.

[33] Noah Wardrip-Fruin. What hypertext is. *HYPERTEXT '04 Proceedings of the fifteenth ACM conference on Hypertext and hypermedia*, págs. 126–127, 2004.