

COMUNIDAD WEB CACHE

KEVIN MORAGA



Un Modelo de Caché Web Distribuido Sobre el protocolo HTTP y P2P

*Proyecto para optar por el grado de
Maestría Científica en Telemática*

Programa de Maestría
Escuela de Ingeniería en Computación
Instituto Tecnológico de Costa Rica



Noviembre, 2014 – Versión 1.0

Kevin Moraga: *Comunidad Web Caché*, Un Modelo de Caché Web Distribuido Sobre el protocolo HTTP y P2P, © Noviembre, 2014

PROFESOR ASESOR:
Isaac Ramírez

LUGAR:
San José, Costa Rica

FECHA:
Noviembre, 2014

Sólo un tonto no tiene *miedo*.
El *valor* es ver el miedo y seguir adelante de todas formas.

— Julian Assange

Dedicado a mis padres, abuela, hermanos y hermanas que han
estado siempre a mi lado.

RESUMEN

La Comunidad Web Caché trata de establecer una comunidad en la cual cada miembro comparta un poco de sus recursos computacionales como almacenamiento, procesamiento y ancho de banda para buscar el bien común. Esto se traduce en un incremento en la eficiencia y la velocidad del servidor web que sirve archivos. Los principales protocolos utilizados en el proyecto son HTTP y P2P. Y es con la combinación de ambos, más la especificación de un nuevo protocolo, que se logra el diseño de la CWC. El diseño toma en cuenta mecanismos de consistencia de datos, de distribución equitativa de trabajo a través de la comunidad virtual y de recuperación de fallos, para darle continuidad al negocio.

*We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.*

— Donald E. Knuth [13]

AGRADECIMIENTOS

Les agradezco a todas aquellas personas que han dedicado un poco de su tiempo, invirtiéndolo en mi formación y la de muchos otros alumnos de este respetado Instituto.

En torno al conocimiento y aporte de ideas, le agradezco a Nereo Campos, Herson Esquivel, Juan Carlos Brenes.

En apoyo relacionado a temas de la tesis, Wathsala Vithanage, Nuwan Gunaratne y Nishshanka Sirisena ¹, quienes fueron muy amables en responder todas las dudas que tuve en relación a su proyecto y a toda L^AT_EX-community por el soporte, ideas y el grandioso software.

¹ Miembros de diseño de Dalesa Cache

ÍNDICE GENERAL

i	INTRODUCCIÓN	1
1	INTRODUCCIÓN	3
1.1	Comunidad Web Caché	5
1.2	Definición del problema	8
1.3	Justificación	10
1.3.1	Innovación	10
1.3.2	Impacto	12
1.3.3	Profundidad	12
1.4	Hipótesis	13
1.5	Objetivos	13
1.5.1	Objetivo Principal	13
1.5.2	Objetivos Específicos	13
1.6	Alcances	14
1.7	Entregables	14
ii	MARCO TEÓRICO	17
2	PROTOCOLO HTTP	19
2.1	¿Cómo trabaja?	19
2.2	Mensajes HTTP	20
2.3	Petición	21
2.4	Respuesta	23
3	PROTOCOLO P2P	27
3.1	Clasificación	27
3.1.1	Directorio Centralizado	27
3.1.2	Directorio Distribuido	28
3.1.3	Directorio Híbrido	28
3.2	Mecanismos de búsqueda en Redes P2P	29
4	MODELOS DE WEB CACHÉ	31
4.1	Web Caching mediante Proxy	32
4.1.1	Proxy Simple	32
4.1.2	Proxy Jerárquico	32
4.1.3	Proxy Descentralizado	33
4.2	Redes CDN	33
5	MODELOS DE CONSISTENCIA	35
5.1	Razones para Replicar	35
5.2	Modelos de Consistencia sin Sincronización	35
5.3	Modelos de Consistencia con Sincronización	37
6	COMUNIDADES VIRTUALES	39
6.1	Definición de una Comunidad Virtual	39
6.2	Características de una Comunidad Virtual	39
6.3	Tipos de Comunidades Vituales	40

iii	DISEÑO DE LA CWC	45
7	ESPECIFICACIÓN DE REQUERIMIENTOS	47
7.1	Actores	47
7.2	Requerimientos Funcionales del Protocolo	47
7.2.1	HTTP SEND Cache	47
7.2.2	HTTP GET Distribuido	48
7.2.3	HTTP SEND Distribuido	48
7.3	Requerimientos Funcionales del Servidor Web Caché	49
7.3.1	Administración de Contenido	49
7.3.2	Administración de Miembros de la Comunidad	50
7.3.3	Implementación de Modelo de Consistencia	51
7.4	Requerimientos Funcionales del Cliente Web Caché	52
7.4.1	Administración de Caché	52
7.4.2	Administración de la velocidad de conexión	52
7.4.3	Modos de Operación de la Caché	52
8	ARQUITECTURA DEL CWC	53
8.1	CWC Server	54
8.1.1	Data Access	54
8.1.2	CWC Data Management	55
8.1.3	CWC Core	56
8.1.4	CWC Caches Catalog	56
8.1.5	CWC Protocol Interpreter	57
8.1.6	CWC Web Interface	57
8.2	CWC Client	57
8.2.1	Data Access	58
8.2.2	CWC Data Management	58
8.2.3	CWC Client Core	60
8.2.4	CWC Cache Catalog	60
8.2.5	CWC HTTP Interpreter	60
8.2.6	CWC Web Browser Interface	60
9	DISEÑO DEL PROTOCOLO CWC	61
9.1	Mensajes HTTP del Protocolo CWC	61
9.1.1	DGET	61
9.1.2	DSEND	62
9.2	Estructura de los Mensajes CWC	63
9.3	Operaciones del Protocolo CWC	63
9.3.1	ACK	64
9.3.2	GetClientId	64
9.3.3	UpdateMembership	64
9.3.4	GetObjectInfo	65
9.3.5	ModifyObject	66
9.3.6	Enable-Disable Object	66
9.3.7	DeleteObject	67
9.3.8	GetMemberStatus	67
9.3.9	ChangeMemberStatus	68

9.3.10	GetListOfFiles	68
9.3.11	GetStats	68
9.3.12	SendStats	69
9.3.13	GetTopCachesbyObject	69
9.3.14	GetConfParams	70
9.3.15	IsAlive	70
9.4	Seguimiento de los miembros conectados	71
9.5	Recuperación de fallas	71
9.5.1	¿Cómo descubrir una falla?	72
9.5.2	Falla del Servidor	72
9.5.3	Falla un Cliente	73
9.6	Implementación del Modelo de Consistencia	73
10	DISEÑO DEL CWC	75
10.1	CWC Server	75
10.1.1	Administración de Objetos en la Caché local	75
10.1.2	Administración del CWC	77
10.1.3	Estadísticas	78
10.1.4	Segmentación de Archivos	80
10.1.5	Administración Remota de Caché	80
10.2	CWC Client	81
10.2.1	Asociación a la comunidad	81
10.2.2	Administración de la Caché Local	82
10.2.3	Otras Funciones de Administración	82
iv	ANÁLISIS Y RESULTADOS	83
11	RENDIMIENTO	85
11.1	Herramientas	85
11.1.1	httpperf	85
11.1.2	OpenWebLoad	85
11.1.3	Time	86
11.1.4	JMeter	86
11.2	Parámetros de las pruebas	87
11.2.1	Servidor Web y Servidor Cache	87
11.2.2	Archivos para Realizar la Prueba	87
11.2.3	Estrategia de las pruebas	88
11.3	Escenarios	88
12	ANÁLISIS Y RESULTADOS	89
12.1	Prototipo CWC	89
12.2	Resultados de Rendimiento	89
12.3	Análisis de rendimiento de velocidad del entorno de CWC	89
12.4	Resultados de Costo	90
12.5	Análisis de costos del Entorno de CW	90
v	CONCLUSIONES	91
13	CONCLUSIONES Y TRABAJO FUTURO	93
13.1	Conclusiones	93

13.2 Trabajo Futuro	94
13.3 Exclusiones	94

BIBLIOGRAFÍA	95
--------------	----

ÍNDICE DE FIGURAS

Figura 1	Ejemplo de Conexión de Video	7
Figura 2	Conexión HTTP	19
Figura 3	Directorio Centralizado	28
Figura 4	Solución Distribuida	29
Figura 5	Directorio Híbrido	29
Figura 6	Conexión Directa	32
Figura 7	Proxy Simple	32
Figura 8	Proxy Jerárquico	33
Figura 9	Proxy Descentralizado	34
Figura 10	Red CDN	34
Figura 11	Comunidad Web Caché	53
Figura 12	Comunicación del CWC Server	54
Figura 13	Arquitectura CWC Server	55
Figura 14	Comunicación del CWC Client	58
Figura 15	Arquitectura CWC Client	59
Figura 16	Funcionamiento del DGET	62
Figura 17	Estructura de un mensaje	63
Figura 18	Funcionamiento de httpperf	85
Figura 19	Funcionamiento de OpenWebLoad	86
Figura 20	Funcionamiento de time	86
Figura 21	Resultados de Tiempo de Respuesta	87
Figura 22	Prototipo de CWC	89

ÍNDICE DE TABLAS

Tabla 1	Uso de Internet por Región Geográfica ITU.	8
Tabla 2	Métodos de HTTP 22	
Tabla 3	Métodos de Petición 24	
Tabla 4	Resultados del escenario Sin Web Caché	89
Tabla 5	Resultados del escenario Prototipo CWC	90
Tabla 6	Comparación de costos 90	

ÍNDICE DE LISTADOS

Listado 1	Tipos de mensajes HTTP	20
Listado 2	Mensaje de petición	21
Listado 3	Línea de petición	21
Listado 4	URI-Solicitada	23
Listado 5	Mensaje de respuesta HTTP	23
Listado 6	Línea de Estado	25
Listado 7	Mensaje ACK	64
Listado 8	Mensaje GetClientId	64
Listado 9	Mensaje de Respuesta GetClientId	64
Listado 10	Mensaje de UpdateMembership	64
Listado 11	Mensaje de GetObjectInfo	65
Listado 12	Mensaje de Respuesta de GetObjectInfo	65
Listado 13	Mensaje de ModifyObject	66
Listado 14	Mensaje de EnableObject	66
Listado 15	Mensaje de DeleteObject	67
Listado 16	Mensaje de GetMemberStatus	67
Listado 17	Mensaje de Respuesta de GetMemberStatus	67
Listado 18	Mensaje de ChangeMemberStatus	68
Listado 19	Mensaje de GetListOfFiles	68
Listado 20	Mensaje de Respuesta de GetListOfFiles	68
Listado 21	Mensaje de GetStats	69
Listado 22	Mensaje de Respuesta de GetStats	69
Listado 23	Mensaje de SendStats	69
Listado 24	Mensaje de GetTopCachesbyObject	69
Listado 25	Mensaje de Respuesta de GetTopCachesbyObject	70
Listado 26	Mensaje de GetConfParams	70
Listado 27	Mensaje de Respuesta de GetConfParams	70
Listado 28	Mensaje de IsAlive	70

ACRÓNIMOS

API	Application Programming Interface
BGP	Border Gateway Protocol
CV	Comunidad Virtual
CWC	Comunidad Web Caché
DNS	Domain Name Service
HTTP	Hypertext Transfer Protocol
NLB	Network Load Balancing
P2P	Peer to Peer
UML	Unified Modeling Language
URI	Uniform Resource Identifier
WAN	Wide Area Network
WWW	World Wide Web

Parte I

INTRODUCCIÓN

En la siguiente sección se describirán las razones que impulsaron el desarrollo de ésta investigación, mostrando el problema, su justificación y además cuáles son los objetivos propuestos.

INTRODUCCIÓN

Durante los últimos años el incremento en el uso de Internet ha sido, se podría decir, exponencial. Diariamente nacen cientos de sitios web ofreciendo información de todos los posibles temas, desde el estado del clima hasta la imagen más reciente de nuestro vecino planeta rojo. El acceso a Internet pasó de estar en unas cuantas manos, a ser usado diariamente por la mayoría de las personas en el mundo.

Uno de los principales aportes, que ayudó sin duda a la consolidación del Internet como fenómeno mundial, en la década de los noventas, fue la creación del protocolo HTTP, el cual vino a establecer una forma universal para intercambiar información. Este protocolo utiliza una arquitectura cliente-servidor, donde un cliente realiza una petición de algún recurso al servidor y el servidor satisface esta petición mediante el envío del recurso solicitado.

Una consecuencia directa de la aceptación del protocolo HTTP en el mercado mundial fue el crecimiento de Internet. Pero esto también implicó una marcada evolución en la tecnología, pues actualmente se puede acceder a gran cantidad de información instantáneamente y la mayoría de los usuarios de Internet ignoran todo lo que esto implica. Detrás de este uso intensivo se esconde, una enmarañada red tejida de conexiones redundantes, encargadas de transportar las conexiones desde los usuarios hasta los servidores que poseen la información.

Por otro lado, es posible dividir Internet en dos mundos totalmente diferenciados, el primero desde su concepción, hecha en ARPANET, que fue con la idea principal de soportar cualquier desventura, y su fin es el transporte de información (capa de transporte). Y el otro mundo reside en el contenido per sé, el cual corresponde precisamente a la capa de aplicación, donde el protocolo HTTP nuevamente se hace presente como uno de los principales mecanismos de acceso a la información.

Ahora bien ¿Qué justifica esta evolución en la tecnología tanto desde el punto de vista de protocolo como de dispositivos involucrados? La respuesta es muy sencilla. Al ser HTTP un protocolo tan versátil, un lenguaje universal para el intercambio de datos, se comenzó a utilizar para diferentes aplicaciones. Se pasó de un simple intercambio de información, a ser la piedra angular por la cual se mueve la mayoría de las actividades humanas. Hoy en día tenemos muchos usos, con la aparición de la multimedia los requerimientos de todas las partes involucradas (cliente, servidor y medio de comunicación) se han

HTTP: Hypertext Transfer Protocol. Este protocolo se define por primera vez en el RFC 1945 [3].

ARPANET fue una de las primeras redes de conmutación de paquetes operacional producto de la guerra fría, cuando Estados Unidos quería una red de comando y control que pudiera sobrevivir una guerra nuclear [24].

incrementado. Actualmente se comparten videos de gran tamaño, se transmite televisión en alta definición, se comparte música, documentos; en fin un gran número de archivos que utilizan grandes anchos de banda, que hasta hace unos años para una conexión de 56Kbps parecía imposible.

Pero ¿Acaso esta maravilla tecnológica es gratuita? y la respuesta es no. Como se mencionó anteriormente cada parte involucrada requiere adquirir una nueva tecnología. Y es por ello que se analizará cada una de las partes involucradas en el intercambio de información, por separado.

En el caso del cliente, es posible imaginar a una persona en su casa que renta una conexión a Internet de banda ancha para compartir archivos, revisar su correo electrónico, escuchar música, ver películas, interactuar en algún juego en línea, participar en video conferencias; y es que con estos requerimientos el equipo requerido es bastante costoso, pero se puede decir que no inaccesible.

Ahora en el otro extremo, el servidor. Por ejemplo se supondrá un servidor que aloja un sitio web que hace transmisión de videos, los cuales son aportados por los mismos usuarios del servicio web. Ahora bien, si en este ejemplo el sitio recibiera un gran número de visitas, 100 visitas por segundo, usted como lector y experto encontraría una solución fácil al problema que supone el alto tráfico; y aumentaría el número de servidores que atienden el servicio web, además al esquema de solución se le podría agregar un balanceo de carga. Con estos previos ajustes, en efecto se puede hacer frente a este gran número de usuarios.

NLB, Network Load Balancing. Consiste en el balanceo de tráfico a través de dos o más enlaces WAN sin la utilización de complejos algoritmos de enrutamiento como BGP [29].

Si bien es cierto, la anterior solución es válida y en muchas ocasiones implementada. Es necesario tomar en cuenta otras variables y cuantificar todos los costos asociados, por ejemplo el precio por servidor, la solución de NLB y una conexión de Internet lo suficientemente buena como para servir videos. Unas cuantas sumas hasta aquí ya arrojarían números bastante elevados, pero lo más preocupante es que nunca es suficiente, siempre es necesario estar actualizándose, pues lo único constante es que la tecnología es cambiante y las necesidades del cliente van en aumento.

Por otro lado, cubrir esta necesidad es una cosa, pero además de esto hay que pensar en otros riesgos asociados y que son inherentes en la publicación de un sitio en la Web por ejemplo ataques por Denegaciones de Servicio Distribuidas (DDoS), una caída del proveedor de servicios o simplemente un incremento inesperado de usuarios. Son muchas las variables que cubrir, y en todas ellas se podría decir que existen soluciones ya conocidas, pero todas implican mayor inversión por parte del individuo que desea publicar el contenido.

Debido al problema expuesto anteriormente, aparece como solución el Caché. En su definición más sencilla podemos decir que es:

"una región de memoria rápida que contiene copias de datos. El acceso a la copia en caché es más eficiente que el acceso a la original" [23].

En este caso en particular se podría ver como un modelo mediante el cual, objetos que se encuentran hospedados en un sitio web son almacenados en otros servidores de forma que pueden ser distribuidos desde otros servidores distintos al servidor web original, esto reduce el número de peticiones que llegan al servidor original lo que disminuye su carga (en el marco teórico de este documento se repasarán los modelos de web caché existentes).

El caché ha sido un tema de estudio en cientos de tesis e investigaciones, pero las soluciones usadas actualmente se limitan a soluciones que deben ser adquiridas por el dueño del sitio web y como se ha discutido anteriormente, esto no es muy rentable, o bien se podrían utilizar soluciones que sean adquiridas por el cliente en su beneficio individual. Parece ser un callejón sin salida, donde la única solución es la inversión en nuevo equipo.

Por otro lado, hay otros dos aspectos de suma importancia que fueron traídos con el auge del Internet, el primero es el establecimiento de comunidades virtuales, en las cuales un grupo de Internautas se reúnen alrededor de un sitio web que presenta información de su interés y definidos como comunidad ayudan para mantenerlo, bajo un conjunto de reglas propias.

El segundo aspecto es el compartir archivos en Internet (tal vez el más polémico de todos) y el que más le interesa a la presente investigación; es el protocolo P2P. Éste básicamente permite que un número de usuarios que cuentan con un archivo específico lo compartan con otros de manera simultánea. Esto quiere decir que si 100 usuarios tienen un archivo de interés común y éste tiene un tamaño de 100MB, en lugar de descargarlo desde un solo servidor, descarga pequeñas partes desde cada uno de los usuarios que lo poseen, haciendo el proceso de descarga es mucho más rápido y evitando que se sature a un solo servidor.

Para ir finalizando esta introducción, el presente documento pretende proponer una posible solución de bajo costo para resolver el problema anteriormente expuesto. A esta solución se le llamó Comunidad Web Caché.

Caché: "Es una región de memoria rápida que contiene copias de datos. El acceso a la copia en caché es más eficiente que el acceso a la original" [23].

1.1 COMUNIDAD WEB CACHÉ

La idea de la Comunidad Web Caché nace a partir de las siguientes premisas:

1. Los usuarios cuentan con una gran cantidad de recursos en sus computadoras, los cuales son desperdiciados en más de un 80 %.
2. La mayoría del contenido en Internet es de interés común y no solo de la persona que lo publica, entonces si éste es de interés común ¿Porqué solo unos cuantos deben mantenerlo?. ¿Porqué no se puede distribuir entre todos?.
3. Quien publica contenido en Internet no siempre tiene la capacidad para invertir en grandes servidores que puedan servir a un gran número de usuarios.
4. Así como existen comunidades en Internet que se conforman por un interés común, por ejemplo hacer amigos, deportes, entre otros. ¿Porqué no crear una comunidad alrededor de un sitio web?.

La propuesta que se quiere implementar es dejar de lado el modelo tradicional de servir archivos mediante HTTP e implementar un nuevo modelo de Web Caché distinto al tradicional, utilizando las ventajas de ambos, combinado con la creación de comunidades de virtuales y la utilización de los recursos de los miembros de éstas comunidades; es posible utilizar un protocolo de transferencia distribuida de contenido, en este caso P2P, para servir archivos desde diferentes localizaciones.

Basado en la figura 1 se supondrá el siguiente ejemplo: se cuenta con el sitio www.cwc.org que sirve archivos de video de más de 30Mb, con un modelo tradicional HTTP (cliente-servidor) un cliente obtendría el archivo desde un único servidor. Ahora bien, si se tienen 1000 clientes realizando esta misma operación, serían muchas solicitudes para el servidor (en el caso que éste tuviera recursos limitados), por lo tanto es necesario analizar el aumento de los recursos del servidor, entre otras cosas.

El presente proyecto consiste en crear una comunidad alrededor de sitio anteriormente mencionado. En caso que al menos 30 de esos 1000 clientes formaran parte de la comunidad y que éstos tienen los recursos suficientes para servir archivos, entonces es posible servir los archivos en cuestión desde 31 lugares diferentes, en lugar que desde un único punto. Es evidente las ventajas que se pueden dar con un modelo como este. Ahora ¿Qué pasa si miembros de la comunidad no tienen suficientes recursos? En lugar de servir todo el archivo desde una sola localización, ese archivo se puede partir y servir desde múltiples ubicaciones, osea, que en lugar de servir archivos grandes de 30MB o más, se pueden servir sólo una porción de éstos. Esto siguiendo el concepto de una comunidad donde cada miembro aporta lo que puede, en beneficio de todos.

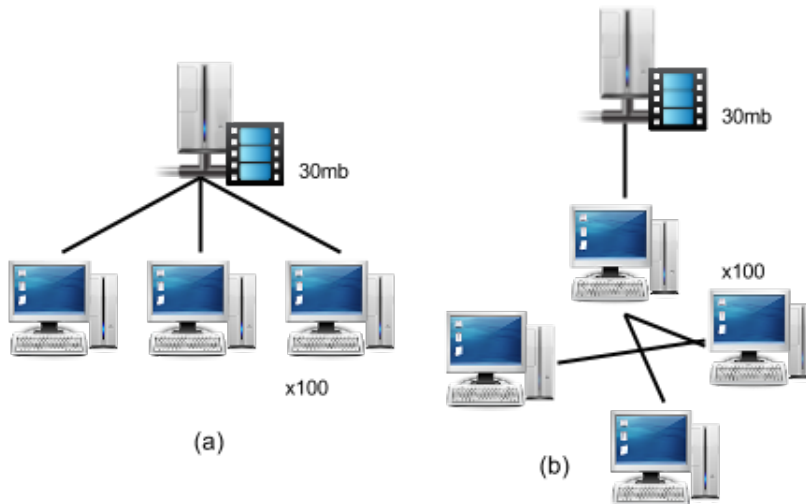


Figura 1: Ejemplo de Conexión de Video

Otra idea podría ser que los clientes con menos recursos sirvan archivos pequeños como por ejemplo imágenes. Esto le quitaría gran cantidad de trabajo al servidor ya que una imagen significa un hilo de ejecución más que se debe crear para servir un archivo.

Uno de los objetivos es que estas comunidades se conformen de manera automática y voluntaria, esto quiere decir que un usuario instala un complemento en su navegador web y el sitio al que trata de acceder (el cual utiliza el protocolo de CWC) realiza el intercambio de manera automática sobre los archivos más relevantes que se desean mantener en caché, haciendo que éstos archivos puedan ser servidos desde la nueva ubicación dada por el nuevo miembro de la comunidad.

La cache podrá tener dos modos de operación, el primero consiste en la administración por parte del usuario, esto quiere decir que cuando el usuario accede a un contenido este queda automáticamente en la cache y el usuario puede decidir si dejarlo o eliminarlo. El segundo sería administrado por el servidor, o sea que cuando este modo se encuentra activo el servidor tendrá derecho a utilizar los recursos asignados a la caché, por lo que el servidor podrá enviar archivos que considere que deberían de estar en la caché o borrar los que no considere necesarios.

Se ampliará con mayor detalle el funcionamiento y diseño del protocolo de CWC en los siguientes capítulos.

1.2 DEFINICIÓN DEL PROBLEMA

En los últimos años se ha dado un alto crecimiento en el número de usuarios de la "World Wide Web", por ejemplo a continuación se muestran las estadísticas del crecimiento de usuarios de acuerdo a el área geográfica en los últimos 10 años:

REGIÓN	2005	2010	2014
África	2.4 %	9.8 %	19 %
Américas	35.9 %	50.5 %	65.5 %
Estados Árabes	8.3 %	23.0 %	40.6 %
Asia y Pacífico	9.4 %	22.5 %	34.4 %
Europa	46.3 %	66.6 %	74.8 %

Tabla 1: Uso de Internet por Región Geográfica [ITU](#).

Como se puede ver en la tabla 1, en casi diez años se ha duplicado o más la cantidad de usuarios de Internet, lo cual implica que se dará una mayor demanda sobre los sitios web, esto se traduce en que el dueño de un sitio web que presta un servicio deberá hacer una inversión en más recursos computacionales, como por ejemplo poder procesamiento y ancho de banda para poder servir a sus usuarios de la mejor manera posible.

Esto quiere decir que la publicación de un sitio web con contenido de calidad no es tan trivial (se hablará en términos de contenido de calidad, ya que el contenido justificará la cantidad de usuarios que recibirá un sitio web, o sea entre mayor sea la calidad del contenido ofrecido por un sitio web para un grupo de usuarios en particular, así será la cantidad de usuarios que recibirá) ya que se deberán tomar en cuenta aspectos de crecimiento en el número de usuarios que visiten el sitio web y los requerimientos computacionales necesarios para poder prestar un servicio de calidad, acorde con los contenidos publicados.

Tomando en cuenta que realizar una publicación en Internet implica un costo para el publicador y éste se incrementa conforme aumenta el número de usuarios que soliciten los contenidos del sitio web, se podría elaborar una función de costo donde se incluyan las variables anteriormente expresadas en función de una calidad de servicio aceptable.

Este costo se traduce en que los interesados en publicar, deban de pagar por conexiones a Internet de Alta Velocidad y adquirir grandes granjas de servidores que soporten la infraestructura para hacer frente a la demanda. Es por ello que en la actualidad, solo las grandes compañías son las únicas con la capacidad de sufragar estos gastos,

mediante el establecimiento de grandes caches regionales que puedan atender el tráfico generado en un área específica.

Ahora ¿Qué pasa con las pequeñas compañías, instituciones sin fines de lucro e individuos que quieren crear un sitio web con contenido de calidad y no cuentan con suficientes recursos para sufragar los gastos generados por un alto número de usuarios?. Una opción es buscar financiamiento o poner precio a sus contenidos o simplemente buscar la forma de ganar dinero mediante el bombardeo de publicidad en los sitios web, lo cual es un aspecto negativo para los usuarios.

Una vez que se han visto las necesidades de recursos computacionales para publicar contenido en Internet que es accedido por un alto número de usuarios, ¿Qué pasa con los recursos computacionales que tiene el usuario/cliente?, ¿Los recursos del cliente tienen una utilización de un 100 % en todo momento o se encuentran ociosos la mayoría del tiempo?, ya se notó que la cantidad de usuarios con acceso al World Wide Web crece enormemente, esto implica que un gran número de usuarios cuentan con una conexión a Internet con una velocidad aceptable para acceder a los contenidos de cotidianos (como videos, música, entre otros). Además también implica que cada uno de esos usuarios cuenta con una o más computadoras, las cuales obviamente cuentan con recursos computacionales aceptables para poder visualizar los contenidos ofrecidos por los sitios web, entonces se tendría la siguiente pregunta: ¿Qué sucede con todos éstos recursos de cómputo (limitado al ancho de banda, procesamiento y memoria) mientras se accede a alguna comunidad virtual, se utiliza una aplicación de ofimática, se reproduce algún contenido de multimedia en Internet o simplemente se ejecuta una aplicación de chat modo texto? Y la respuesta es simple: se desperdician los recursos, están ociosos y están esperando a ser utilizados.

En términos de definición de la problemática, éste se encuentra compuesto por cinco partes esenciales:

1. Falta de recursos de quien publica contenido de calidad en Internet, esto implica un alto costo para poder atender un mayor número de clientes.
2. Un alto desperdicio de recursos en los clientes, entre estos recursos podemos citar: Ancho de Banda, Almacenamiento, Memoria y Procesamiento.
3. Una baja en la calidad del servicio conforme aumenta el número de clientes, tiempos de respuesta poco aceptables o bien se ve comprometida la continuidad del negocio.
4. El número de usuarios en Internet se incrementa constantemente cada año, para muestra la tabla 1, la cual reúne el crecimiento en los últimos 10 años.

5. El problema inherente de tener un sitio con una disponibilidad del 100 %, soportando las solicitudes usuales de los clientes, las denegaciones distribuidas y las fallas en los servicios.

¿Cómo dar solución a este problema?, ¿Es posible que la solución se encuentre en una utilización eficiente de los recursos tanto del publicador cómo de los clientes?, ¿Es posible encontrar una solución de bajo costo que aproveche las causas de estos problemas para dar una solución que resuelva los problemas?, ¿Es posible el establecimiento de una comunidad virtual en la cual cada miembro de la misma, aporte un poco de recursos computacionales con el afán de mantener contenido de calidad en Internet, el cual pueda ser accedido de una forma rápida y fiable? Y aun más importante ¿Es posible establecer una comunidad virtual basada en una aplicación de bajo costo que utilice de una manera eficiente los recursos computacionales aportados por cada miembro y además se pueda utilizar con las aplicaciones de navegadores y servidores web utilizados actualmente?

1.3 JUSTIFICACIÓN

1.3.1 *Innovación*

La Comunidad Web Caché trata de establecer una comunidad en la cual cada miembro comparta un poco de sus recursos computacionales como almacenamiento, procesamiento y ancho de banda para buscar el bien común. Esto se traduce en un incremento en la eficiencia y la velocidad del servidor web que sirve archivos.

Por otro lado, se han identificado varias problemáticas que se deben tomar en cuenta en la creación del presente proyecto y que es necesario cubrir en torno al diseño de la Comunidad Web Caché. Entre ellas se pueden mencionar:

1. El incremento exponencial en el número de usuarios en Internet.
2. El incremento en los recursos computacionales accesibles a los usuarios en Internet, los cuales son desperdiciados.
3. La falta de recursos computacionales por parte de quien publica contenido de calidad en Internet, cuando aumenta el número de usuarios se vuelve un problema ya que entre más usuarios se necesitan más recursos para poder servir a estos usuarios de una manera satisfactoria.
4. La tendencia en los contenidos de moda en Internet, archivos estáticos de gran tamaño básicamente contenidos de multimedia.

5. El gran numero de comunidades con objetivos específicos que se establecen en Internet de la cual se puede tomar la decisión de formar parte.
6. La necesidad de presentar continuidad del negocio, tomando en cuenta los ataques de DDoS y tolerancia a desastres.

El proyecto pretende la utilización de tecnologías y productos ya establecidos, entre los cuales se tienen:

1. Protocolo HTTP
2. Protocolo P2P
3. Navegadores Web
4. Servidores Web

Se podría resumir el funcionamiento de la Comunidad Web Caché como:

Quien publica contenido de calidad en Internet podrá decidir cual contenido podrá ser mantenido por la comunidad (el contenido que podrá ser mantenido por la comunidad es el que se pueda mantener en la caché, esto por el hecho de respetar el copyright). Los miembros de la comunidad podrán aportar un poco de sus recursos computacionales; es importante que ellos decidan la cantidad que aportarán. Cuando un miembro de la comunidad solicita un archivo, como un video, este será servido desde el servidor principal y desde los otros miembros de la comunidad que posean este archivo, sincronizado y previamente transferido mediante un mecanismo de P2P. Lo anterior supone que debería de incrementar la velocidad al momento de responder la solicitud, pues el archivo es seccionado y transmitido desde distintas ubicaciones, con distintos recursos y en el peor de los casos se volvería al modelo tradicional donde el archivo es distribuido desde el servidor principal.

Uno de los puntos focales del CWC es precisamente la parte comunitaria y es por ello que en el diseño se tomarán en cuenta varios conceptos propios de comunidades virtuales como:

1. Aporte de recursos o participación libre: Los miembros de la comunidad decidirán qué cantidad de recursos computacionales aportaran.
2. Adhesión abierta: Puedo decidir formar parte de la comunidad o no.

3. Respeto por los contenidos privados: Se respeta el Copyright, quien publica los contenidos de calidad podrá decidir si estos pueden ser mantenidos por la comunidad o no.
4. Bien común: los contenidos deben ser servidos de manera rápida y eficiente.
5. Apoyo: Si alguno de los miembros de la comunidad falla otro podrá tomar su lugar (tolerancia a fallos).

En general, el CWC busca el establecimiento de una comunidad cooperativa sobre Internet utilizando los navegadores y servidores web actuales, y se diferencia de otras soluciones ya existentes. Por ejemplo, los Proxys buscan mejorar el rendimiento sobre una red de área local.

Por otro lado, uno de los modelos más parecidos a CWC son los Proxy Federados, mejor conocidos como "Cooperative Web Caching". Éstos buscan tener un conjunto de servidores proxy que se comuniquen y que compartan los contenidos que tienen en cache sobre cualquier tipo de red, podrían utilizar en una WAN o incluso en Internet. Pero en comparación a éste último, en el CWC cualquier usuario puede formar parte de la comunidad, sin necesidad de tener que configurar un Proxy dedicado.

1.3.2 *Impacto*

El proyecto impactará tanto al publicador de contenido de calidad en Internet, así como a los usuarios, en los siguientes aspectos:

PUBLICADOR

- Conforme el número de miembros de la comunidad crezca, se podrá experimentar menos carga en los servidores, por consiguiente se podrán servir muchos más usuarios.

USUARIO

- Conforme el número de miembros de la comunidad crezca, se podrá experimentar una utilización más eficiente de sus recursos computacionales, una mejora en los tiempos de respuesta cuando se hace una petición a un servidor web y una mayor tolerancia a fallos, dándole continuidad al negocio.

1.3.3 *Profundidad*

En relación a los temas que cubrirá el presente proyecto son:

- Se realizará una investigación del estado del arte, en relación a los protocolos, algoritmos y mecanismo que ayuden a soportar el diseño del CWC.
- Se realizará un levantamiento de requerimientos relacionado a las necesidades entorno a CWC.
- Se creará un diseño completo de la implementación de CWC.
- Se realizarán modelos que prueben el rendimiento y costos de la CWC.

Quedan fuera del alcance de este proyecto:

- La aplicación del protocolo de CWC en un ambiente de páginas dinámicas.
- La aplicación del protocolo de CWC en un ambiente de flujo de video o sonido.
- La seguridad en general, tanto del servidor como del cliente. No se especificó ningún mecanismo que permita la comprobación y la autenticidad de los elementos.

1.4 HIPÓTESIS

El desarrollo de una caché web distribuida que permita compartir de manera transparente y eficiente los recursos computacionales distribuidos ofrecidos colaborativamente mediante una comunidad de servidores, tendrá un impacto positivo en el bajar los costos y mejorar la continuidad del negocio para las comunidades virtuales.

1.5 OBJETIVOS

1.5.1 *Objetivo Principal*

Desarrollar una caché web distribuida que permita compartir de manera transparente y eficiente los recursos computacionales distribuidos ofrecidos colaborativamente mediante una comunidad de servidores.

1.5.2 *Objetivos Específicos*

- Compilar y comparar trabajos relevantes relacionados con sistemas distribuidos
- Examinar las oportunidades de mejoramiento estratégico de las plataformas existentes.

- Determinar los requerimientos de la caché web distribuida basada en HTTP.
- Diseñar una arquitectura de una caché web distribuida bajo un conjunto de principios tales como: transparencia de ubicación de los recursos, solución de bajo costo y calidad de servicio.
- Desarrollar un prototipo de la caché web distribuida.
- Analizar el rendimiento de velocidad y el costo del entorno de CWC.

1.6 ALCANCES

- Investigación del funcionamiento de Protocolos P2P y HTTP.
- Investigación de los diferentes modelos de caching existentes con el fin de identificar sus características, arquitectura, funcionamiento, debilidades y fortalezas. Esto ayudará en una etapa posterior para realizar el diseño de la Comunidad Web Caché.
- Investigación sobre el establecimiento de comunidades virtuales y las implicaciones de estas.
- Investigación a soluciones que utilicen los diferentes modelos de web caching, esto permitirá tener un punto de comparación para nuestros modelo y establecerá los umbrales a mejorar.
- Investigación de modelos de consistencia.
- Elaboración del diseño del CWC el análisis de las investigaciones previas. Este modelo se limitará únicamente a contenido estático y no se desarrollará el diseño para la compatibilidad de contenido dinámico y contenidos de flujo de video o sonido, los cuales implican modelos de consistencia mucho más complejos, además de algoritmos de reemplazo y muchas otras implicaciones de diseño que necesitan de una investigación aún mayor.
- Elaboración de prototipo de sistema para compartir archivos de manera distribuida tomando en cuenta ciertas partes del diseño. Este prototipo pretende convertirse en la base para el desarrollo del CWC.

1.7 ENTREGABLES

1. Informe investigación funcionamiento de protocolo HTTP.
2. Informe investigación funcionamiento de protocolo P2P.
3. Informe investigación de modelos de Web Caché existentes.
4. Informe investigación sobre comunidades virtuales.

5. Informe investigación de aplicaciones que utilicen los diferentes modelos de Web Caché.
6. Informe investigación de modelos de consistencia.
7. Documento de requerimientos de Comunidad Web Caché.
8. Prototipo de CWC Cliente.
9. Prototipo de CWC Server.
10. Análisis de rendimiento de velocidad del entorno de CWC.
11. Análisis de costos del Entorno de CWC.

Parte II

MARCO TEÓRICO

En los siguientes capítulos se aclararán temas relacionados a los protocolos utilizados por CWC, como lo son HTTP y P2P. Además establecerán todas aquellas definiciones necesarias para el desarrollo de la presente investigación.

PROTOCOLO HTTP

2.1 ¿CÓMO TRABAJA?

El protocolo HTTP es un protocolo petición/respuesta. Un cliente envía una petición al servidor en la forma de un método de petición, URI, versión de protocolo, seguido de un mensaje estilo- MIME conteniendo los modificadores de la petición, información del cliente y posiblemente el contenido del cuerpo sobre una conexión con el servidor, el servidor resuelve con un mensaje de estatus, incluyendo la versión del protocolo del mensaje y el código de error o éxito, seguido de un mensaje estilo-MIME que contiene la información del servidor, la meta-información de la entidad y posiblemente el contenido del cuerpo de la entidad. La conexión más simple sería cuando el agente de usuario realice una conexión directa con el servidor de origen. Un ejemplo de la conexión de una petición se muestra en la figura 2.

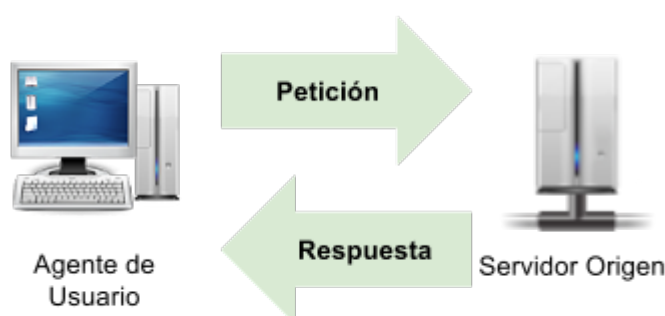


Figura 2: Conexión HTTP

En conexiones más complejas pueden intervenir uno o más intermediarios, como lo pueden ser:

- Proxies que son agentes de reenvío, este recibe peticiones para un URI en su forma absoluta, sobre escribe toda o parte del mensaje y reenvía las peticiones re-formateadas hacia el servidor identificado por la URI.
- Gateway es un agente de recepción, actuando como una capa sobre otros servidores y si es necesario traduce la petición al protocolo del servidor subyacente.

- Túnel: Actúa como un punto de transmisión entre dos conexiones sin cambiar el mensaje, los túneles son usados cuando la conexión pasa a través de un intermediario (como un firewall) aún cuando el intermediario no entiende los contenidos del mensaje.

2.2 MENSAJES HTTP

TIPOS DE MENSAJES: Los tipos de mensajes HTTP son de petición de un cliente hacia el servidor y de respuesta de un servidor hacia el cliente.

```
"HTTP-message = Request | Response ; HTTP/1.1 messages"
```

Listado 1: Tipos de mensajes HTTP

Los mensajes Request y Response utilizan la forma genérica de mensajes, definido en RFC822 para transferencia de mensajes. Ambos mensajes consisten una línea de inicio, uno o más valores de encabezado, una línea vacía que indica el final de los valores de encabezado y posiblemente un cuerpo del mensaje [6].

A continuación se describirán con más detalle los mensajes mínimos necesarios para establecer una comunicación HTTP:

ENCABEZADOS DEL MENSAJE: Los valores HTTP, los cuales incluyen general-header, request-header, response-header y entity-header, siguen el mismo formato genérico del RFC822. Cada campo del encabezado consiste de un nombre seguido por ":" y un valor. Los nombres de campos son case-insensitive, el valor se puede extender múltiples líneas y el orden en que se reciben de los campos del header no es significativo [6].

CUERPO DEL MENSAJE: El cuerpo del mensaje de un mensaje HTTP es utilizado para transportar el cuerpo de la entidad asociado con la respuesta o la petición. El cuerpo del mensaje difiere del cuerpo de la entidad solo cuando una codificación de transferencia ha sido aplicada. "message-body = entity-body | entity-body codificado" El campo Transfer-Encoding debe ser utilizado para indicar cualquier codificación de transferencia aplicada por una aplicación para asegurar la transferencia adecuada del mensaje. Transfer-Encoding es una propiedad del mensaje no de la entidad [6].

LARGO DEL MENSAJE: El transfer-length de un mensaje es el largo del cuerpo del mensaje a como este aparece en el mensaje, sin que se haya aplicado ningún filtro. Cuando el cuerpo del mensaje es incluido con un mensaje, el transfer-length de ese cuerpo es determinado por uno de [6]:

- Cualquier mensaje de respuesta que no incluya cuerpo del mensaje como las repuestas de estado 1xx, 204 y 304 son siempre terminadas con la primer línea vacía después de los campos de encabezado presentes en el mensaje.
- Si un campo de Transfer-Encoding se encuentra especificado y tiene cualquier valor diferente a "Identity", entonces el transfer-length está determinado por el uso de "chunked" transfer-coding, a menos que el mensaje se terminara por que se cerró la conexión.
- Si un campo Content-Length se encuentra presente, su valor decimal en octetos representa el entity-length y transfer-length. Content-Length no se debe enviar si los dos anteriores no son iguales.
- Si el mensaje utiliza el tipo de media "multipart/byteranges" y el largo de transferencia no está especificado, entonces este tipo de media determina el transfer-length. Este tipo de media no debe ser utilizado a menos que quien envía conozca que el recipiente lo puede parsear.
- Por un cierre de conexión por parte del servidor

2.3 PETICIÓN

Un mensaje de petición de un cliente hacia un servidor incluye, en la primera línea de ese mensaje, el método a ser aplicado al recurso, el identificador del recurso y la versión del protocolo en uso [6].

```
"Request = Request-Line * (( general-header | \\  
request-header | entity-header ) CRLF) \\  
CRLF  
[message-body]"
```

Listado 2: Mensaje de petición

LÍNEA DE PETICIÓN: La línea de petición inicia con un token de método, seguido por la URI que se está pidiendo y la versión del protocolo y terminando con CRLF (retorno de carro y cambio de línea). Los elementos son separados por caracteres SP [6].

```
"Request-Line = Method SP Request-URI SP \\  
HTTP-Version CRLF"
```

Listado 3: Línea de petición

MÉTODO El token de método indica el método a ser aplicado al recurso identificado por la URI-pedida. El método es case-sensitive. El listado de los métodos del protocolo HTTP se puede observar en la tabla 2.

MÉTODO	DESCRIPCIÓN
OPTIONS	Representa una solicitud de información acerca de las opciones de comunicación disponibles en la cadena de Petición/Respuesta identificada por el URI-pedida.
GET	Significa que se recupere cualquier información (en la forma de entidad) que es identificada por la URI-pedida.
HEAD	Es idéntica a GET con la excepción de que el servidor no retornara en cuerpo del mensaje en la respuesta.
POST	Se utiliza para solicitar que el servidor original acepte la entidad envuelta en la petición como un nuevo subordinado del recurso identificado por el URI-pedida en la línea de petición.
PUT	Solicita que la entidad envuelta sea almacenada bajo el URI-pedida suplida.
DELETE	Solicita que el servidor original borre el recurso identificado por el URI-pedida
TRACE	Es utilizada para invocar un remoto, de nivel de aplicación loop- back del mensaje de petición.
CONNECT	Se utiliza con proxies que pueden dinámicamente cambiarse para ser un túnel.
extension-method	Definidos por alguna aplicación.

Tabla 2: Métodos de HTTP [Tanenbaum and Wetherall](#).

URI-SOLICITADA La URI-Solicitada es un identificador uniforme de recurso (URI) e identifica el recurso sobre el cual se va a aplicar el mensaje de petición.

```
"Request-URI = "*" | absoluteURI | abs_path | authority"
```

Listado 4: URI-Solicitada

Las opciones del URI-pedida dependen de la naturaleza de la petición. El asterisco significa que la petición no aplica a ningún recurso en particular, pero al servidor en sí mismo y solo es permitido cuando el método utilizado no necesariamente aplica a un recurso [6].

EL RECURSO IDENTIFICADO POR UNA PETICIÓN Para extraer el identificador de recurso por una petición en Internet es determinado mediante el proceso de examinar ambos URI-pedida y el campo del encabezado HOST. Un servidor de origen que no diferencie recursos de acuerdo en el host solicitado debe utilizar las siguientes reglas para identificar el recurso pedido [6]:

1. Si URI-pedida es un URI absoluta, el HOST es parte de URI-pedida. Cualquier campo de encabezado HOST debe ser ignorado.
2. Si URI-pedida no es una URI absoluta y la petición incluye un campo de encabezado HOST, el HOST es determinado por el campo HOST de la petición.
3. Si el HOST identificado por las reglas 1 y 2 no es un HOST valido en el servidor entonces el servidor deberá devolver la respuesta 400 Bad Request.

CAMPOS DE ENCABEZADO DE UNA PETICIÓN El encabezado de la petición permite a el cliente pasar información adicional acerca de la petición y acerca del cliente al servidor. Estos campos actúan como modificadores de la petición, con semántica equivalente a los parámetros en la invocación de métodos en los lenguajes de programación. Un ejemplo de los campos de encabezado más importantes se pueden ver en la tabla 3.

2.4 RESPUESTA

Después de recibir el interpretar un mensaje de petición, el servidor responde con un mensaje de respuesta HTTP.

```
"Response = Status-Line *(( general-header | \\  
response-header | entity-header ) CRLF)  
CRLF  
[message-body ]"
```

Listado 5: Mensaje de respuesta HTTP

MÉTODO	DESCRIPCIÓN
Accept	Se utiliza para especificar ciertos tipos de media que son aceptables por la respuesta.
Accept-Charset	Se utiliza para indicar cuales CHARSETS son aceptables en la respuesta.
Accept-Encoding	Similar a ACCEPT pero limita los content-encodings aceptables en la respuesta.
Accept-Language	Similar a ACCEPT pero con la diferencia que restringe el conjunto de lenguajes que son preferidos como respuesta a la petición.
Authorization	Se utiliza cuando el agente de usuario quiere identificarse con él con el servidor. Se envían las credenciales.
Cache-Control	Se utiliza para especificar directivas que deben ser obedecidas por todos los mecanismos de cache a lo largo de la cadena de Petición/Respuesta.
Cookie	Muestra la cookie establecida privamente que se regresa al servidor.
Host	Contiene el nombre de DNS del servidor.
User-Agent	Contiene la información sobre el navegador y su plataforma.
Referer	El URL anterior desde el cual provino la solicitud.
Set-Cookie	La cookie que debe guardar el cliente.
Server	Información sobre el servidor web.

Tabla 3: Métodos de Petición [Tanenbaum and Wetherall](#).

LÍNEA DE ESTADO: La primera línea de un mensaje de respuesta es Status-Line, la cual se encuentra conformada por una versión de protocolo seguida por un código numérico de estatus y su frase de texto asociada, con cada elemento separado por caracteres SP [6].

```
"Status-Line = HTTP-Version SP Status-Code \\  
SP Reason-Phrase CRLF"
```

Listado 6: Línea de Estado

CÓDIGO DE ESTADO Y FRASE: El código de estado es un código de resultado de 3 dígitos que indica el intento de entender y satisfacer la petición. Esos códigos ya se encuentran definidos. La frase intenta dar una corta descripción textual del Código de estado [6].

ENCABEZADO DE RESPUESTA: Los campos del encabezado de respuesta permiten al servidor pasar información adicional acerca de la respuesta que no pueden ser colocadas en la Status-Line. Estos campos del encabezado dan información acerca del servidor y acerca de futuros accesos al recurso identificado por la URI-Pedida [6].

PROTOCOLO P2P

Una red peer-to-peer (P2P) o red de pares, es una red de computadoras en la que todos o algunos aspectos de ésta, funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí. Es decir, actúan simultáneamente como clientes y servidores respecto a los demás nodos de la red [30].

Las redes peer-to-peer aprovechan, administran y optimizan el uso del ancho de banda de los demás usuarios de la red por medio de la conectividad entre los mismos, obteniendo más rendimiento en las conexiones y transferencias que con algunos métodos centralizados convencionales, donde una cantidad relativamente pequeña de servidores provee el total del ancho de banda y recursos compartidos para un servicio o aplicación. Alfredo F. nos dice que de forma simple puede verse como la comunicación entre pares o iguales utilizando un sistema de intercambio [5].

Los sistemas compañero a compañero (o P2P por sus siglas en inglés) permiten que computadoras de usuario final se conecten directamente para formar comunidades, cuya finalidad sea el compartir recursos y servicios computacionales. En este modelo, se toma ventaja de recursos existentes en los extremos de la red, tales como tiempo de CPU y espacio de almacenamiento. Las primeras aplicaciones emergentes se orientaban a compartir archivos y a la mensajería [5].

3.1 CLASIFICACIÓN

3.1.1 Directorio Centralizado

Esta primera visión se puso en práctica para el año 1997 con la red Napster [19]. Bajo este modelo los clientes o peers realizan el descubrimiento y búsqueda de los otros miembros mediante un servidor central, luego el mismo servidor se encarga de negociar la conexión entre ambos clientes.

Este tipo de red P2P se basa en una arquitectura monolítica en la que todas las transacciones se hacen a través de un único servidor que sirve de punto de enlace entre dos nodos y que, a la vez, almacena y distribuye los nodos donde se almacenan los contenidos. Poseen una administración muy dinámica y una disposición más permanente de contenido. Sin embargo, está muy limitada en la privacidad de los usuarios y en la falta de escalabilidad de un sólo servidor, además

de ofrecer problemas en puntos únicos de fallo, situaciones legales y enormes costos en el mantenimiento así como el consumo de ancho de banda [30].

El principal problema presentado en este modelo es que el servidor P2P Central es un posible punto de quiebre que pueda atentar contra la estabilidad de la red.

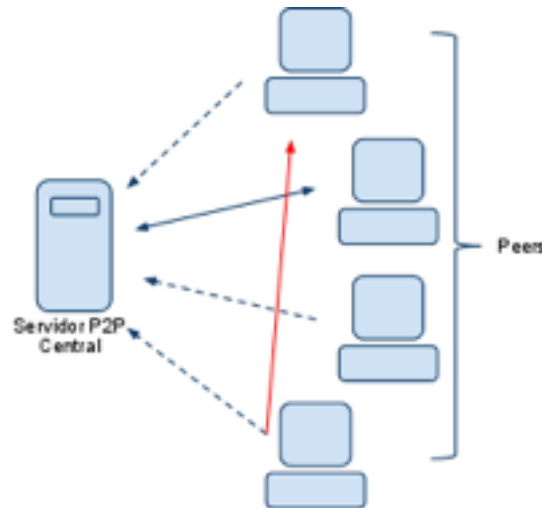


Figura 3: Directorio Centralizado

3.1.2 Directorio Distribuido

Uno de los ejemplos más conocidos de este tipo de redes es Freenet [19]. Las redes P2P de este tipo son las más comunes, siendo las más versátiles al no requerir de un gestionamiento central de ningún tipo, lo que permite una reducción de la necesidad de usar un servidor central, por lo que se opta por los mismos usuarios como nodos de esas conexiones y también como almacenistas de esa información. En otras palabras, todas las comunicaciones son directamente de usuario a usuario con ayuda de un nodo (que es otro usuario) quien permite enlazar esas comunicaciones [30].

3.1.3 Directorio Híbrido

Un ejemplo de este tipo de redes es Gnutella 0.6 [22]. Este tipo de red se encuentra formada por una estructura jerárquica la cual comprende nodos de usuario y ultrapeers. Cuando un nuevo nodo entra al sistema, este se transforma como un nodo usuario quedando en el perímetro de la red, donde éste no posea ninguna responsabilidad de enrutamiento y su única función es la de enviar consultas a los ultrapeers.

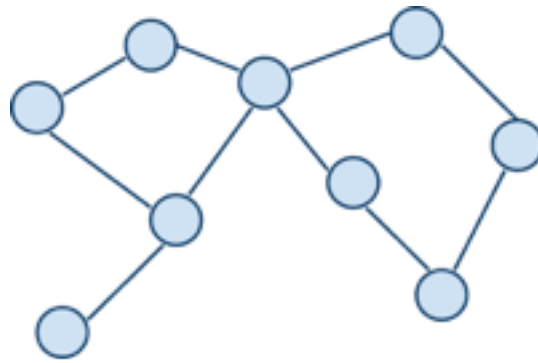


Figura 4: Solución Distribuida

Por otro lado nodos con capacidades más elevadas y con habilidades de procesamiento mejoradas, son promovidos como *ultrapeers*. Éstos son los responsables de realizar enrutamiento y de resolver consultas de los nodos usuario [22].

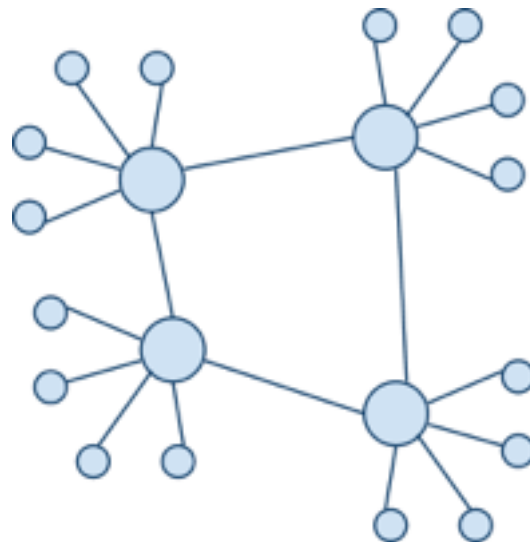


Figura 5: Directorio Híbrido

3.2 MECANISMOS DE BÚSQUEDA EN REDES P2P

Las redes P2P son clasificadas a su vez en: Redes no estructuradas y Redes estructuradas. Algunos han aseverado que éstos algoritmos son alternativas que compiten entre sí [19].

REDES NO ESTRUCTURADAS: Las redes No Estructuradas han sido llamadas redes de "primera generación". Los nodos en estas redes se unen conectándose a través de cualquier otro nodo. En las primeras versiones de Gnutella, ésta se podía clasificar como No Estructurada [19]. En estas redes el costo por enrutamiento debido a difusión o *broadcasting* es muy elevado o también pue-

de ocasionar que hayan fallos encontrando el recurso solicitado.
[19]

REDES ESTRUCTURADAS: Las redes estructuradas tienen importantes ventajas. Proporcionan un mecanismo determinista de localización, de tal forma que se asegura que un recurso será encontrado siempre que esté en la red. Además, los mensajes de búsqueda son encaminados de forma eficiente, de tal forma que el recurso es encontrado en $O(\log n)$ saltos, donde n es el tamaño de la red [30]. Estas redes han sufrido dos críticas principalmente [19]. La primera de ellas es entorno a la transitoriedad de un nodo, que al final afecta qué tan robusto es la red. La otra crítica es debido a que éstas redes no soportan búsquedas por palabra clave ni consultas complejas como los sistemas No Estructurados.

El web caché no es un concepto nuevo. Este tema ya se ha estudiado y también ha sido objeto de numerosos papers, artículos y tesis. Por otro lado el concepto de caché web distribuida es un poco más nuevo pero también suma bastantes proyectos entorno a él.

Como primer acercamiento, es necesario definir el significado de caché web (en ocasiones también llamado proxy) [27]:

"Se llama caché web a la caché que almacena documentos web (es decir, páginas, imágenes, entre otros) para reducir el ancho de banda consumido, la carga de los servidores y el retardo en la descarga. Un caché web almacena copias de los documentos que pasan por él, de forma que subsiguientes peticiones pueden ser respondidas por el propio caché, si se cumplen ciertas condiciones."

En resumen, se podría entender una caché web como una forma de almacenar temporalmente documentos que son concurrentemente solicitado con el fin de reducir el ancho de banda saliente. Por otro lado, la presente investigación se centra en modelos que comparten dinámicamente éstos documentos.

Un concepto muy parecido es el de caché web cooperativo. Según Rowstron A. [20] mediante esta técnica, el almacenamiento dedicado a la caché de una serie de servidores-proxy adyacentes se gestiona conjuntamente como si se tratase de una única cache distribuida. Las políticas de gestión de esta cache se realiza teniendo en cuenta el estado y los contenidos de todos los servidores-proxy que la componen. Por ejemplo, las políticas de reemplazo en la cache cooperativa tendrán en cuenta las estadísticas de acceso en todos los servidores-proxy cooperantes.

El caché web cooperativo permite incrementar considerablemente el tamaño de la caché, pero a costa de reducir la probabilidad de acierto local. Bajo éste esquema, un único servidor-proxy no dispone de los contenidos más populares, provocando un aumento del porcentaje de peticiones que se tienen que servir remotamente desde los otros servidores que componen la cache ó desde el servidor principal.

Basado en la investigación del estado del arte, se han podido identificar proyectos que giran alrededor de la presente tesis. Entre ellos están: CoralCDN [7], Transparent Distributed Web Caching [15], Dalesa [25] y Squirrel [20].

4.1 WEB CACHING MEDIANTE PROXY

4.1.1 *Proxy Simple*

Primero se mostrará el modelo tradicional de cliente-servidor. Bajo este modelo el cliente tiene una conexión directa con el servidor y no existe caché en medio, por lo que en dicha conexión se obtienen los servicios bajo demanda, haciendo uso únicamente del caché local del explorador web.



Figura 6: Conexión Directa

La siguiente figura 7 muestra el típico modelo de proxy, donde existe un servidor central por el cual los clientes transitan y le solicitan el recurso deseado. El servidor proxy descarga el recurso solicitado, lo almacena en su caché y por último lo reenvía al solicitante. Si otro cliente solicitara el mismo recurso, el servidor le respondería con el recurso almacenado localmente (si la caché haya caducado aún).

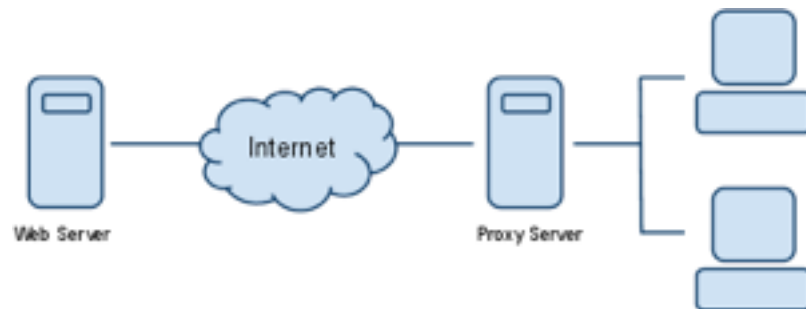


Figura 7: Proxy Simple

4.1.2 *Proxy Jerárquico*

La figura 8 muestra el modelo de proxy jerárquico o también conocido como web caché cooperativo. En este modelo se tienen servidores proxy ubicados de una manera jerárquica en una Red de área local. Cuando un cliente solicita un recurso, éste lo hace a su proxy jerárquicamente más cercano. Si dicho servidor posee los recursos éste se lo reenvía al cliente. En caso que no posee el recurso, el proxy le reenvía la solicitud de su cliente al servidor proxy de nivel superior. Si el servidor de nivel superior posee el recurso lo reenvía al servidor proxy

que solicitó el recurso, éste último se deja una copia en su caché local y reenvía el recurso al cliente original.

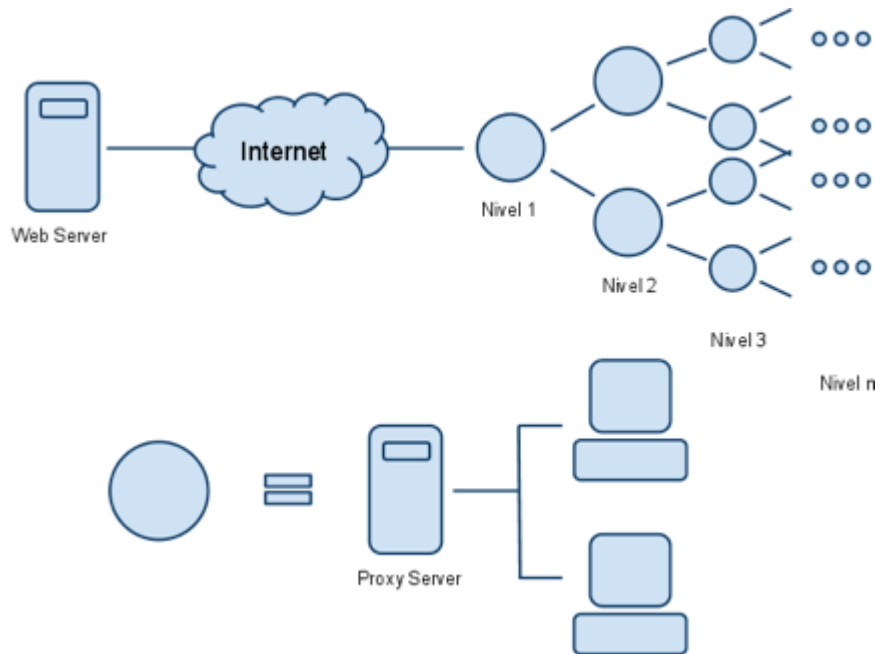


Figura 8: Proxy Jerárquico

4.1.3 Proxy Descentralizado

Bajo este modelo se encuentran los proyectos de Squirrel y Dalesa. Estos proyectos intentan utilizar los conceptos de P2P para implementar y compartir una cache distribuida a través de una LAN. Como se muestra en la figura 9. Cada computadora forma parte del proxy descentralizado, y utilizando un software creado por los respectivos participantes del proyecto, pueden compartir la cache. Este modelo es el que más se acerca al modelo que se propone como tesis aunque todavía dista en ciertas características.

4.2 REDES CDN

Una red de distribución de contenidos es una red de equipos que opera de forma transparente para entregar contenido de sus clientes cumpliendo principalmente alguno de los siguientes objetivos:

1. Escalabilidad
2. Coste
3. Eficiencia

Haciendo una definición más amplia una Red CDN es un sistema de computadoras el cual contiene copias de datos, ubicados en varios

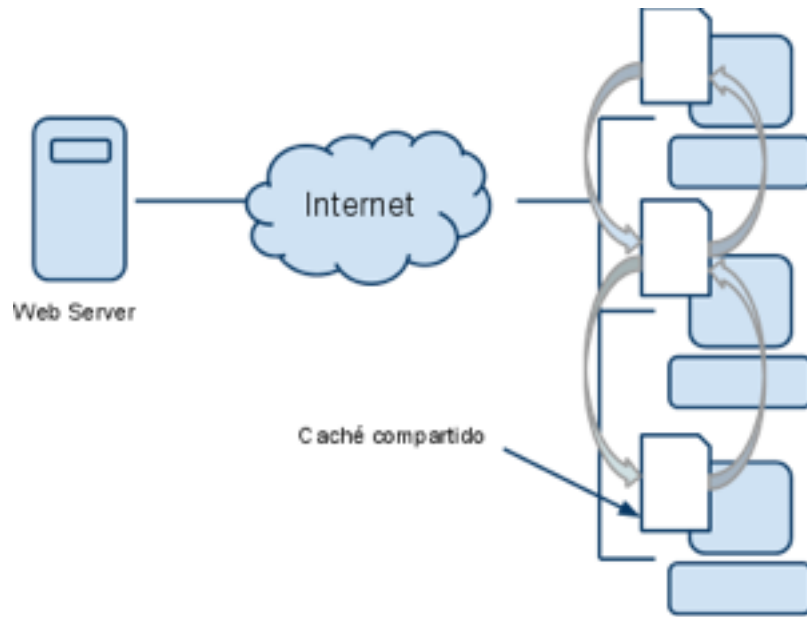


Figura 9: Proxy Descentralizado

puntos de la red para así maximizar el ancho de banda para el acceso de los datos desde los clientes a través de la red. Un cliente accede una copia de los datos, la más cercana, contrario al concepto de que todos los clientes accedan al mismo servidor central provocando un cuello de botella en el servidor [28].

El contenido puede incluir objetos web, objetos descargables (archivos de medios, software, documentos, entre otros), aplicaciones, media streams en tiempo real, y otros componentes.

En la siguiente figura se puede observar el funcionamiento de una Red CDN distribuida a través de Internet.

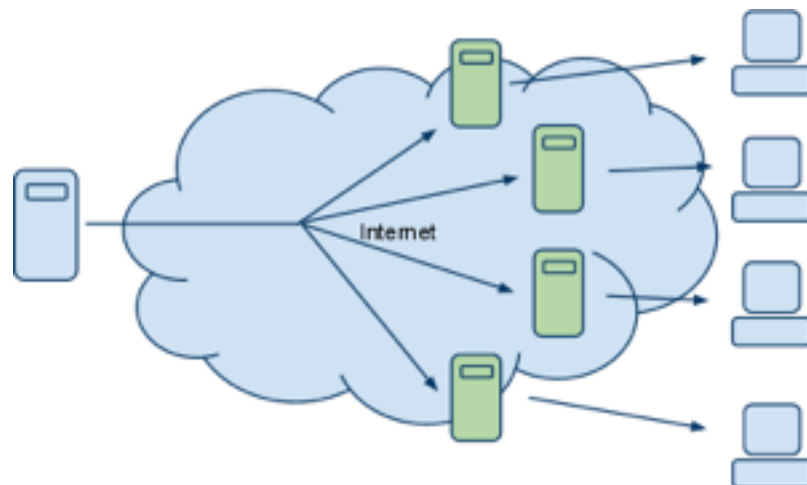


Figura 10: Red CDN

MODELOS DE CONSISTENCIA

Un requerimiento importante en los sistemas distribuidos es la replicación de datos, lo datos por lo general se replican para mejorar fiabilidad y incrementar el rendimiento, uno de los problemas más importantes es, mantener todas las réplicas consistentes, esto quiere decir que cuando una copia se actualiza, necesitamos asegurar que las otras copias se actualizan también.

5.1 RAZONES PARA REPLICAR

1. Los datos se replican para incrementar la fiabilidad del sistema, esto quiere decir que si el sistema ha sido replicado este puede seguir trabajando después de alguno de sus nodos falle, mediante el cambio a uno que si funcione. Al mantener muchas copias es posible proporcionar una mejor protección contra datos corruptos.
2. Los datos se replican para incrementar el rendimiento, es importante cuando un sistema distribuido necesita ser escalado en números y áreas geográficas. En el caso de números, cuando un sistema necesita procesar mucha información, este se escala agregando otro nodo al sistema y replicando los datos, para dividir el trabajo. En el caso de área geográfica se coloca una copia cerca del proceso que lo necesita, para reducir el tiempo de acceso a los datos.

5.2 MODELOS DE CONSISTENCIA SIN SINCRONIZACIÓN

LINEARIZABILITY O CONSISTENCIA ESTRICTA: Según Herlihy M. [10] linearizability es la condición correcta para objetos concurrentes. Las operaciones write que se ejecutan en cualquiera de los sitios de un sistema, se ejecutan simultáneamente en todas las copias de los datos, osea es la condición ideal siempre las copias son coherentes. En pocas palabras es la consistencia más estricta, esta establece que cualquier operación de lectura en el sistema, debe devolver el valor con la última actualización antes de realizar la lectura. Lo cual quiere decir que el tiempo máximo para propagar una actualización al resto de sitios en el sistema es cero.

CONSISTENCIA SECUENCIAL: Es un modelo de consistencia un poco más débil, que la consistencia estricta. Básicamente Lamport [14] lo define mediante:

"El resultado de cualquier ejecución es el mismo que si las operaciones de todos los procesos fueran ejecutadas en algún orden secuencial, y las operaciones de cada proceso individual aparecen en esta secuencia en el orden especificado por su programa".

La consistencia secuencial no garantiza que una operación de lectura devuelva el valor escrito por otro proceso. Lo único que asegura es que todos los sitios vean todas las operaciones en el mismo orden.

CONSISTENCIA CAUSAL: Según Reynal M. [18], en un sistema se da consistencia causal si las escrituras parcialmente relacionadas de forma causal (sobre el mismo objeto concurrente) son vistas por todos los procesos en el mismo orden, las escrituras concurrentes pueden ser vistas en un orden diferente en máquinas diferentes.

Este modelo es aun más débil que la consistencia secuencial [1], la cual requiere que todos sitios vean las escrituras en el mismo orden.

Cuando un sitio realiza una lectura seguida de una escritura, aun en diferentes objetos concurrentes, se dice que la primera operación esta causalmente ordenada antes que la segunda, debido a que el valor almacenado por la escritura puede haber sido dependiente del resultado de la lectura. De manera similar una operación de lectura puede estar causalmente ordenada después de una escritura previa en el mismo objeto concurrente que almacena lo que fue leído. De la misma manera dos operaciones de escritura en el mismo sitio se encuentran causalmente ordenadas en el mismo orden que fueron realizadas.

Si existen operaciones que no están causalmente relacionadas se dice que son concurrentes.

CONSISTENCIA PRAM: Lipton y Sandberg [16] definieron el modelo de consistencia Pipelined RAM (PRAM) y comúnmente conocido como Consistencia FIFO. Este modelo es aun más débil que la consistencia causal, esta consiste en eliminar el requisito de que las operaciones relacionadas se vean en orden por parte de todos los sitios en un sistema. Para esto las escrituras por un sitio son recibidas por otros procesos en el orden en que son realizadas, pero las escrituras de sitios diferentes pueden verse en un orden distinto en sitios diferentes [17].

Este tipo de consistencia es fácil de implementar, ya que no existe garantía del orden en que los diferentes sitios vean las escrituras, más que dos o más escrituras por parte de un proceso

llegan en orden, como si estuvieran entubadas. Remontándonos a la definición de consistencia causal, podemos decir que en consistencia FIFO todas las escrituras y lecturas locales (realizadas en el mismo sitio) están causalmente relacionadas, mientras que todas las que suceden en sitios diferentes son concurrentes [17].

5.3 MODELOS DE CONSISTENCIA CON SINCRONIZACIÓN

CONSISTENCIA DÉBIL: La consistencia débil utiliza variables de sincronización para propagar las escrituras a todo el sistema, mediante esto una operación de escritura en un sitio, tomará un candado el cual evitará que otros puedan leer o escribir hasta que la operación termine y se propague al resto de los sitios que conforman el sistema. A este mecanismo se le llama exclusión mutua.

El modelo tiene las siguientes condiciones [17]:

- Los accesos a las variables de sincronización son secuencialmente consistentes.
- No se permite ingresar a una variable de sincronización hasta que todas las operaciones de escritura hayan terminado en el resto de los sitios.
- No se permite realizar un acceso a los datos (operaciones de lectura y escritura) hasta realizar todos los accesos a las variables de sincronización.

CONSISTENCIA RELEASE: Es muy parecida a la Consistencia Débil [17], lo que hace es dar solución a un problema de rendimientos que tiene ésta. Cuando se accede a una variable de sincronización, el objeto concurrente no sabe si se debe a que el sitio ha terminado de escribir en el objeto concurrente o está a punto de iniciar su lectura. Para una implementación más eficiente que la Consistencia Débil, se crean dos variables de sincronización que diferencien estos casos. Estas operaciones se llaman release y acquire [8], antes de realizar una operación de escritura a un objeto concurrente un sitio debe adquirir el objeto mediante la operación acquire y más tarde liberarlo mediante release.

Para contar con consistencia release se debe cumplir:

- Antes de realizar una operación a un objeto concurrente, deben terminar con éxito todas las adquisiciones anteriores del sitio en cuestión.
- Antes de realizar un release, se deben terminar todas las escrituras y lectura del sitio.
- Los accesos a release y acquire deben de ser consistentes con el procesador.

CONSISTENCIA DE ENTRADA: El modelo de Consistencia de Entrada es aún más débil que el de Consistencia Release [4]. Utiliza las mismas operaciones de sincronización mencionadas anteriormente *acquire* y *release*. La diferencia radica en que requiere que cada objeto concurrente se asocie con alguna variable de sincronización [17], como en un modelo de exclusión *barrier* o *tree*.

La consistencia de entrada debe cumplir [4]:

- No se permite a unos sitios realizar un *acquire* a un objeto concurrente con respecto a un sitio hasta que se realicen todas las actualizaciones de los datos compartidos protegidos con respecto a este sitio.
- Antes de permitir el acceso de modo exclusivo a una variable de sincronización por un sitio, ningún otro sitio debe poseer la variable de sincronización, ni siquiera en modo no exclusivo.
- Después de realizar un acceso en modo exclusivo a una variable de sincronización, no se puede realizar el siguiente acceso en modo no exclusivo de otro sitio a esta variable de sincronización hasta haber sido realizado con respecto al propietario de esa variable.

COMUNIDADES VIRTUALES

6.1 DEFINICIÓN DE UNA COMUNIDAD VIRTUAL

Las Comunidades Virtuales (CV) son lugares en la web donde las personas se pueden encontrar y "hablar" electrónicamente entre sí. Estas comunidades normalmente están rodeadas por intereses comunes. En un inicio las CV actuaban como si fuera una cafetería en lugar de un asunto meramente comercial, aunque éstas eran por naturaleza comerciales, donde las personas realizaban transacciones y comercio [9].

Las nociones de "comunidad" y "virtual", plantean la tentativa de definir una comunidad virtual como "una congregación de cibernautas que integran una comunidad que aparenta ser real al simular los efectos de las congregaciones sociales humanas reales o tradicionales, pero sin llenar todas las características de estas".

Por otro lado, se podría definir CV como: Se denomina comunidad virtual a aquella comunidad cuyos vínculos, interacciones y relaciones tienen lugar no en un espacio físico sino en un espacio virtual como Internet.

6.2 CARACTERÍSTICAS DE UNA COMUNIDAD VIRTUAL

Según Whittaker [26] una comunidad debe por lo menos tener los siguientes atributos esenciales:

1. Los miembros deben de compartir un mismo objetivo, interés, necesidad o actividad que provea una razón primaria para pertenecer a una comunidad.
2. Los miembros deben de estar comprometidos a una participación activa y en ocasiones interacciones intensas, donde se compartan actividades y sentimientos comunes entre los participantes.
3. Los miembros deben tener acceso a los recursos compartidos y además deben de existir políticas para determinar el acceso a éstos recursos.
4. Debe existir una reciprocidad de la información, soporte y servicios entre los miembros.

5. Compartir un contexto, puede ser social, de lenguaje o simples protocolos.

Por otro lado algunos atributos menos necesarios podrían ser [26]:

1. Roles y reputación diferenciada.
2. Conciencia de los límites de los miembros y de la identidad del grupo.
3. Criterio de iniciación.
4. Un historial de participación.
5. Eventos o rituales
6. Un espacio físico compartido
7. Una membresía voluntaria.

6.3 TIPOS DE COMUNIDADES VIRTUALES

A grandes rasgos podemos clasificar las CV en tres grandes categorías: de ocio, profesionales y de aprendizaje [2].

Por otro lado para Armstrong y Hagel [11] existen dos tipos de Comunidades Virtuales: las orientadas a los usuarios y las orientadas a la organización.

ORIENTADAS A LOS USUARIOS: En las orientadas a los usuarios, son ellos los que definen el tema de la Comunidad y se pueden dividir en:

- *Geográficas*: agrupan personas que viven o que están interesadas en intercambiar Información sobre una misma área geográfica.
- *Temáticas*: orientadas a la discusión de un tema de interés para los usuarios.
- *Demográficas*: reúnen usuarios de características demográficas similares.
- *De ocio y entretenimiento*: dirigidas a aquellos cibernautas que ocupan su tiempo libre en juegos en red. Se crean por tipos de juegos como estratégicos, de simulación, etc.
- *Profesionales*: para aquellos expertos en una materia que desarrollan su actividad concreta en un área profesional definida, generalmente asociada a una formación superior. Especialmente en el caso de las profesiones liberales, cuando se trabaja de manera independiente.
- *Gubernamentales*: Los organismos gubernamentales han creado Comunidades Virtuales a las que puede acudir el ciudadano para informarse y/o discutir.

- *Eclécticas*: son aquellas Comunidades Virtuales mixtas, que intentan un poco de todo: zona de ocio, una vía de transmisión y comportamiento cultural, etc.

ORIENTADAS A LA ORGANIZACIÓN: Por el contrario en las orientadas hacia la organización, el tema es definido según los objetivos y áreas de trabajo de la organización donde reside la comunidad, y podemos dividir las en:

- *Verticales*: que aglutinan a usuarios de empresas de diferentes ramas de actividad económica o a organizaciones institucionales.
- *Funcionales*: referidas a un área específica del funcionamiento de la organización, por ejemplo: mercadeo, producción, relaciones públicas.
- *Geográficas*: que se concentran en una zona geográfica cubierta por la organización.

Por el contrario Salinas [21] agrupa las Comunidades Virtuales utilizando un enfoque más pragmático:

DE DISCURSO: El ser humano es la criatura social y puede hablar cara a cara sobre intereses comunes, pero también puede compartir estos intereses con otros semejantes más lejanos mediante los medios de comunicación.

DE PRÁCTICA: Cuando en la vida real alguien necesita aprender algo, normalmente no abandona su situación normal y dedica su esfuerzo en clases convencionales sino que puede formar grupos de trabajo (comunidades de práctica), asigna roles, enseña y apoya a otros y desarrolla identidades que son definidas por los roles que desempeña en el apoyo al grupo.

DE CONSTRUCCIÓN DE CONOCIMIENTO: El objetivo de este tipo de comunidades es apoyar a los estudiantes a perseguir estratégica y activamente el aprendizaje como una meta.

DE APRENDIZAJE: Si una comunidad es una organización social de personas que comparten conocimiento, valores y metas, las clases como las conocemos no son comunidades ya que los estudiantes están desconectados o están compitiendo unos con otros. Las comunidades de aprendizaje surgen cuando los estudiantes comparten intereses comunes. Las TIC pueden contribuir a conectar alumnos de la misma clase o de alrededor del mundo, con el objeto de lograr objetivos comunes.

Una última clasificación realizada por Cabero [2] apoyado en las conclusiones de Salinas [21]. Identifica una serie de grupos en función de:

MODO DE ASIGNACIÓN: Dado el modo de asignación se pueden encontrar:

- Comunidades de asignación libre por parte de los miembros.
- Comunidades de asignación voluntaria.
- Comunidades de asignación obligatoria.

FUNCIÓN PRIMARIA DE LA COMUNIDAD: Las comunidades también se pueden clasificar por su función:

- *Distribución:* Cuando la principal función de la comunidad radica en la distribución de información o mensajes, entre los miembros.
- *Compartir:* Se trata de comunidades donde prima el intercambio de experiencias y recursos.
- *Creación:* Cuando se generan procesos de trabajo colaborativo con el objeto de lograr materiales, documentos, proyectos compartidos.

GESTIÓN DE LA COMUNIDAD: Por otro lado un punto importante que nos puede ayudar a clasificar una CV es el tipo de gestión que se utilice:

- *Abiertas:* Cuando el acceso (independientemente de la asignación) es abierto y los recursos de la comunidad están a disposición tanto de los miembros como de personas ajenas a la comunidad.
- *Cerradas:* Cuando existe algún procedimiento que impide a las personas ajenas a la comunidad el acceso, de tal forma que los recursos, materiales, producciones, histórico, entre otros; sólo son accesibles para los miembros de la comunidad.

EL OBJETO DE LA COMUNIDAD: En la línea de la clasificación descrita anteriormente, las comunidades virtuales de aprendizaje podemos clasificarlas en función del objeto que persiguen, en:

- Comunidades de aprendizaje propiamente dichas, cuando han sido creadas para que el grupo humano que se incorpora a la comunidad desarrolle procesos de aprendizaje en programas diseñados al efecto.
- Comunidades de práctica, ya definidos anteriormente
- Comunidades de investigación, cuando se trata de comunidades que desarrollando actividades de aprendizaje, el objeto principal es poner en marcha proyectos de investigación conjunta de acuerdo con la filosofía del trabajo cooperativo a través de redes.

- Comunidades de innovación. Similares a las anteriores que buscan compartir, intercambiar y generar procesos de innovación en distintos campos.

Parte III

DISEÑO DE LA CWC

En este apartado de la investigación, se describirá el diseño propuesto para el CWC, reuniendo un conjunto de requerimientos basados en los objetivos de la investigación; además se propondrá un modelo que solucione y sea de ayuda para la comprobación de la hipótesis de ésta investigación.

ESPECIFICACIÓN DE REQUERIMIENTOS

7.1 ACTORES

A continuación se detallará la lista de Actores que interactuarán con CWC.

SERVIDOR WEB El servidor web es el encargado de servir los archivos que son solicitados por el usuario a través de un Agente Cliente (en el RFC de HTTP [6] un Agente Cliente se define como cualquier navegador web utilizado por el usuario), en este caso el sistema interaccionará con el servidor web de manera que las solicitudes que si pueden utilizar el protocolo CWC (Community Web Caching) sean servidas de manera distribuida. En el caso de que una solicitud no pueda utilizar CWC se utilizará el modelo tradicional, uno a uno.

AGENTE CLIENTE El agente cliente es un navegador web, el cual cuando se hace una petición por parte de éste, se verifique si el servidor web puede utilizar el protocolo CWC, si es así entonces al realizar el GET HTTP en lugar de obtener un archivo de una sola localización de obtendrá de múltiples localizaciones o cachés.

USUARIO Entre las actividades que tendrá el usuario tendrá: la instalación del cliente, establecer la velocidad de conexión, iniciar o detener el cliente y administrar los parámetros de la caché local.

ADMINISTRADOR DEL SITIO WEB Entre las actividades que tendrá el administrador: la instalación del cliente, establecer la velocidad de conexión, configurar todos los parámetros del modulo del Servidor Web y Administrara el modulo mediante la consola de administración.

7.2 REQUERIMIENTOS FUNCIONALES DEL PROTOCOLO

7.2.1 HTTP SEND Cache

Este método se deberá implementar para permitir el envío de partes del caché entre la comunidad de caché web. Cuando un cliente envía una petición mediante el HTTP GET Distribuido, cualquier nodo que contenga el caché caliente puede enviar trozos del archivo mediante una respuesta HTTP SEND Cache. Por otro lado, el funcionamiento

de este método se podría interpretar como un *push* hacia los clientes, en caso que éstos se encuentra detrás de un *firewall*.

7.2.2 HTTP GET Distribuido

El cliente debería de reemplazar el método GET común, por uno que permita la obtención de un archivo no sólo del servidor principal, sino también de toda la comunidad de web caching. Este proceso debería de ser transparente y además debería de administrar la descarga desde múltiples ubicaciones; incluyendo manejo de reanudaciones, cambio de servidor en caso que un nodo pierda la conexión, verificación de consistencia del archivo y mecanismos de recuperación de información.

7.2.3 HTTP SEND Distribuido

Este es tal vez el método más importante de CWC. Normalmente cuando un cliente realiza una petición en HTTP esta es servida por el servidor al cual se le hace la solicitud, en otros casos se hace una redirección hacia otro servidor y así sucesivamente. La propuesta es que mediante una comunidad CWC alrededor de un website, cuando se solicita una página la cual tiene múltiples elementos, ya sean de multimedia o bien cualquier cualquier archivo con un tamaño mayor a 1 MB, estos elementos sean servidos no solo por el servidor, sino también por un conjunto de miembros de la comunidad, los cuales tienen copias actualizadas a través de CWC con el funcionamiento de un protocolo P2P. Esto quiere decir, por ejemplo que cuando se sirve un archivo de multimedia, un MPEG de 30 MB, en lugar de que sea obtenido desde solo el servidor, se obtendrá una porción de estos 30 MB desde cada uno de los miembros de la comunidad que tienen este video.

Las ventajas del HTTP Send Distribuido, como se puede concluir es que se reduce el tráfico en el servidor y se hace uso de los recursos computacionales que se encuentran ociosos en los miembros de la comunidad. Con esto se puede tener un servidor web con mucho menos recursos, pero que pueda servir a muchos más clientes y además posee un atributo más: la redundancia la cual es indispensable para la continuidad del negocio. Podríamos perder la capacidad de servir estos archivos por parte de uno de los miembros de la comunidad o incluso del servidor pero se seguiría sirviendo archivos debido a la autonomía de la comunidad.

7.3 REQUERIMIENTOS FUNCIONALES DEL SERVIDOR WEB CACHE

Es necesario que CWCServer sea ejecutado como un módulo de un servidor web existente. Este módulo deberá comunicarse con las cachés por medio de un puerto que deberá ser configurable, por medio de este puerto se intercambiará el protocolo CWC el cual será definido en la sección [?].

El módulo deber contar con una interfaz de consola por medio de la cual el éste pueda ser detenido, reiniciado, habilitado o deshabilitado.

7.3.1 *Administración de Contenido*

Cuando se hace alusión al contenido, se refiere a cualquier objeto que pueda ser servido por medio de CWC, en este caso se debe proveer las siguientes interacciones:

AGREGAR UN CONTENIDO Cuando se agrega un contenido al servidor web, este no es agregado automáticamente a la CWC, por el contrario se debe agregar explícitamente. Esto debido a que puede existir contenido que no pueda ser compartido, ya sea por porque el autor no lo desea, porque sean archivos dinámicos (está fuera del alcance de esta tesis) o porque sean archivos demasiado pequeños los cuales no tienen ningún sentido ser servidos por medio de múltiples clientes. Cuando se agrega un contenido a la CWC se debe generar un suma de verificación (checksum) del objeto, establecer una fecha de expiración y crear las estructuras necesarias para manejar las caches que la tienen.

BORRAR UN CONTENIDO Borrar un contenido, no significa eliminarlo del servidor Web, simplemente que no será servido más por medio de la CWC, esto tiene las siguiente implicaciones: 1) Terminar de servir los clientes que estén tomando el archivo por medio de CWC y 2) Invalidar todas las copias del objeto que ya no va a ser servido mediante CWC.

MODIFICAR CONTENIDO Este tipo de interacción puede suceder cuando se cambie la fecha de expiración, que se modifique el objeto, sea necesario generar de nuevo el checksum o que se habilite/-deshabilite el contenido. Cuando se modifica algún objeto se debe enviar la notificación a las cachés para que lo deshabiliten y lo vuelvan a solicitar.

EXPIRAR CONTENIDO Cada uno de los objetos de contenido tiene asociado un tiempo de expiración, el modulo deberá estar revisando cada cierto tiempo este tiempo de expiración para deshabilitar el contenido que ya no pueda ser servido. Las implicaciones cuando un contenido se expira son: 1) Esperar a que los

clientes que estaban tomando el contenido expirado terminen, 2) Deshabilitar el contenido en la cache, marcándolo como expirado y 3) Enviar el aviso a las cachés que tienen el contenido para que lo deshabiliten y lo borren. Si CWC recibe alguna notificación de alguna de las caches de que se ha borrado algún objeto, deberá de actualizar el objeto indicando que esa cache no tiene una copia del objeto.

ESTADÍSTICAS DE LOS CONTENIDOS Se deberá llevar estadísticas de cuantas veces se ha servido un objeto, desde donde se ha solicitado y cuales cachés lo han proporcionado.

7.3.2 *Administración de Miembros de la Comunidad*

Los miembros de la comunidad son las cachés, es en éste módulo donde se hace la conformación de la comunidad de cachés que tendrán los objetos del sitio web, éstos mismos que se compartirán entre nuevos miembros o miembros que no los posean. Un mayor detalle de las operaciones para el módulo de Administración de Miembros de la Comunidad, son los siguientes:

AGREGAR UN NUEVO MIEMBRO A LA COMUNIDAD La comunidad debe crearse de una manera dinámica y voluntaria. Cuando un usuario cuyo navegador web cuenta con CWCClient, que tiene los recursos computacionales mínimos para formar parte de la comunidad, que solicite un objeto manejado mediante CWC y además desee mantener este objeto en su caché, empezará a formar parte de la comunidad. Esto no quiere decir que un cliente que no cuenta con estas condiciones mínimas no pueda utilizar el protocolo y aprovechar las ventajas de obtener el archivo desde varias localizaciones en lugar de solo una.

También se podrá agregar cachés desde una consola de administración de la CWC, esto con el fin de que el dueño de un Website pueda agregar sus propias caches, ya sea en diferentes localizaciones geográficas o bien como mecanismo de redundancia. Éstas cachés se verán como un servidor que tendrá CWCClient ejecutándose y se le asignarán todos los recursos de la computacionales disponibles. Cuando un nuevo miembro empieza a formar parte de la comunidad, se actualiza la información de los objetos que contiene y puede ser elegido para servir archivos. Cuando se agrega un nuevo miembro se guardará información, como el ancho de banda, el espacio disponible para almacenar, entre otros datos.

MODIFICAR UN MIEMBRO DE LA CACHE En este caso cuando un miembro deja de poseer un objeto o se agrega otro se debe actualizar la información de este, además si se modifican los

recursos computacionales asignados por el cliente a la cache se deberá actualizar esta información.

ELIMINAR UN MIEMBRO DE LA COMUNIDAD Esto se puede realizarse de dos maneras: 1) Que el miembro desinstale el CWC-Client o que se quede sin un objeto de la comunidad a la que pertenece (es necesario recordar que una comunidad se crea alrededor de un sitio web), 2) Que el administrador de la CWC decida eliminar un miembro por alguna razón específica, esta acción se realizaría mediante una consola de administración. Las implicaciones de esta última acción serían: se debe eliminar la cache de la comunidad y además se debe actualizar la información de los objetos que éste contenía.

DESHABILITAR O HABILITAR UNA CACHÉ En el caso de que un cliente apague su computadora, deshabilite la cache en su navegador, se pierda contacto con la caché o se decida deshabilitar por el administrador de la caché; se deberá deshabilitar la caché en cuestión, esta última acción implicaría que si se recibe una petición por un objeto que ésta contiene, ya no será enviado como parte de la lista de caches disponibles. En los casos contrarios se habilitarán.

BOOKKEEPING Se refiere a llevar estadísticas de los miembros de la caché, con el fin de poder sacar un ranking de los participantes. Cuando se recibe una petición, se puede enviar al cliente la lista de los mejores miembros de la comunidad, lo que se traduce en los más confiables, que tienen más o mejores recursos, entre otros.

7.3.3 Implementación de Modelo de Consistencia

El modelo de consistencia es bastante simple, por la naturaleza del HTTP, solo se modifica el contenido por parte del servidor y no por los miembros de la caché, lo que provocaría que las escrituras solo se hagan en un lugar. En este caso la consistencia se limitará a que los miembros de la comunidad revisen, que los objetos que tienen, se encuentren actualizados en relación al servidor, esto con el fin de evitar que los objetos hayan sido dañados por alguna situación externa. Estas situaciones pueden ser un software malicioso, que el mismo sistema operativo allá quedado en un estado inconsistente, entre otras.

Cuando se modifique un objeto, en este caso cuando el objeto propiamente se cambie, se deberá notificar a todos los miembros de la caché, para que éstos invaliden sus copias y vuelvan a solicitarlas.

Esto debería ser suficiente para asegurar la consistencia del contenido en las cachés de todos los miembros de la comunidad.

7.4 REQUERIMIENTOS FUNCIONALES DEL CLIENTE WEB CACHÉ

Este cliente debe de comunicarse con el CWCTServer mediante el protocolo HTTP y registrarse como un nodo más de la red distribuida de cachés.

Además debe de proveer una interfaz donde se pueda administrar el ancho de banda que se destinará para el Comunidad Web Caché. También ésta interfaz deberá de permitir definir el tamaño total del cache local.

7.4.1 *Administración de Caché*

El usuario debe ser capaz de administrar el espacio que tenga asignado para almacenamiento para la caché. Este caché podría estar tanto en memoria o bien en disco, en el caso de que el navegador se cierre.

Además debería de detectar cambios en la página original y actualizar el caché con la información nueva. Debe poder permitir cambiar el tipo de caché, si se hace pasivo o activo. El caché activo debería de obtener las páginas más accedidas en el servidor sin necesidad que el usuario navegue por ellas, mientras que el pasivo solo tendría en caché las páginas visitadas por el usuario.

7.4.2 *Administración de la velocidad de conexión*

El cliente debe de proveer una interfaz donde se pueda establecer la velocidad de conexión destinada a la Comunidad Web Caché. Esta velocidad debería de tener un mínimo.

7.4.3 *Modos de Operación de la Caché*

ADMINISTRADA POR USUARIO Esta modalidad especifica que el usuario cada vez que accede a un contenido, éste quedará dentro de la cache y podrá ser servido. El usuario decidirá si lo mantiene o si lo borra.

ADMINISTRADA POR SERVIDOR En este caso el usuario entra a formar parte de la caché de un sitio en particular, con los parámetros especificados en el cliente CWC. Por ejemplo, dependiendo de cuántos recursos se desea aportar a la caché y el servidor web envía archivos que considere necesiten estar en caché. Por otro lado, el servidor también podrá borrar contenido en la caché.

ARQUITECTURA DEL CWC

El enfoque del CWC consiste en conformar una comunidad alrededor de un sitio web, como cualquier otra comunidad virtual que se crea en Internet.

Por ejemplo, una comunidad que se conforma alrededor de la música de un género, en esta comunidad los usuarios aportan nuevas canciones para ser compartidas por el resto de los usuarios. Es dinámica, los miembros van y vienen.

Es fácil concluir que, una comunidad virtual se reúne alrededor de algún interés en común y ayudan para que esta comunidad se mantenga con el tiempo. La idea es principal del CWC es que los usuarios de un sitio web, se conviertan en pequeñas cachés de éste y de esta manera poder hacer que el sitio web pueda servir a más usuarios.

Como se puede observar en la figura 11, los usuarios de un sitio web específico forman parte de la comunidad de éste, donde aportan recursos que se utilizan para mantener una caché de ciertos archivos, los cuales son servidos desde las cachés en lugar de ser servidos desde el servidor principal.

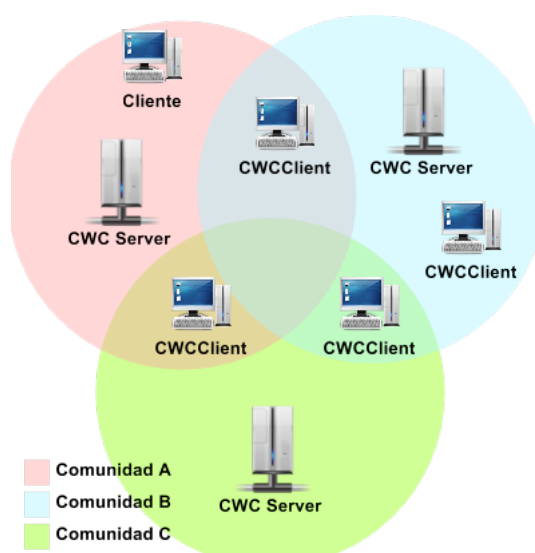


Figura 11: Comunidad Web Caché

En esta sección se describirá la arquitectura que seguirá cada una de las partes que conforman el CWC.

8.1 CWC SERVER

En la figura 12, se puede observar el esquema de una comunicación utilizando CWC, esta comunicación debe ser transparente, de manera que un usuario que no utilice CWC será atendido directamente por el servidor web, en el caso de que un cliente si la utilice entonces esta será atendida por la comunidad y el servidor.

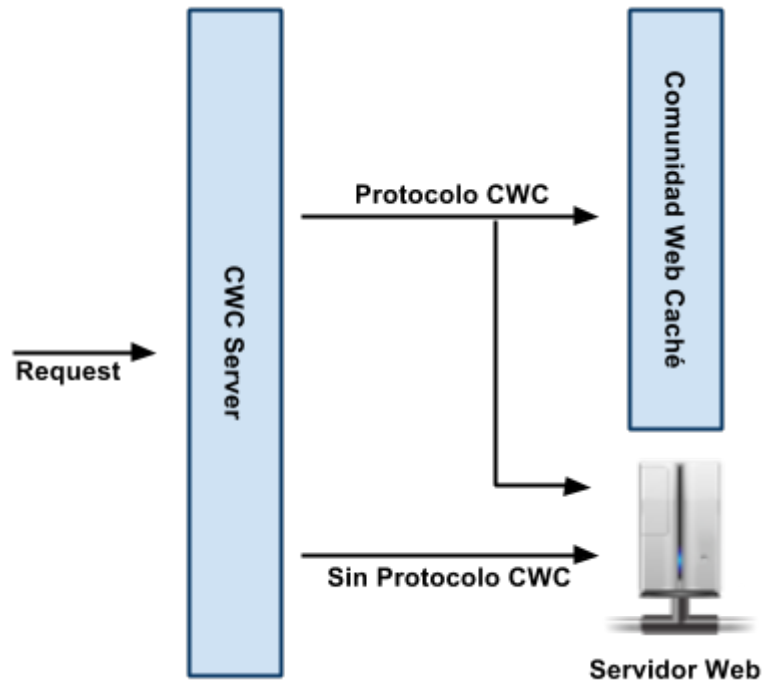


Figura 12: Comunicación del CWC Server

Por otro lado, como se puede observar en la figura 13, se presenta una arquitectura multicapa, en los siguientes apartados se explicará cada una de las capas y la interacción entre ellas.

8.1.1 Data Access

La capa de acceso a datos implementa toda la lógica para realizar el acceso a datos en este caso se deberá implementar para el acceso para:

1. Base de datos: En el caso de la base de datos, este será el repositorio donde se almacenaran las estadísticas y la información de los objetos y los miembros de la CWC.
2. Disco: En este caso se trata de archivos planos de texto, de acceso secuencial que se utilizaran para almacenar logs de acceso, logs del CWC Server en general. Los logs de acceso serán analizados una vez a la semana y se almacenarán en la base de

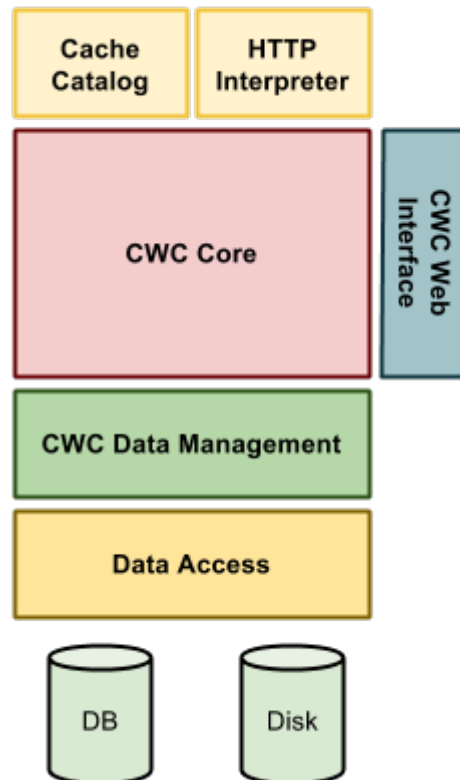


Figura 13: Arquitectura CWC Server

datos. Este proceso no se realiza de forma continua pues es necesario reducir el tráfico al disco y utilizar la mínima cantidad de recursos.

8.1.2 CWC Data Management

El CWC Data Management, actuará como una caché para la base de datos, contendrá la información que se está utilizando de la base de datos actualmente, más la información que se está generando por el uso del sistema. Las estructuras permanecerán en memoria y actuarán como si fuesen la capa de acceso a datos, pero en realidad todo se estará trabajando desde la memoria.

Periódicamente se realizará una descarga de toda esta información hacia la base de datos. La justificación para hacerlo de esta manera es para disminuir la cantidad de accesos a la base de datos y tener la información requerida en memoria para ser accedida de manera rápida y reducir el gasto adicional debido al uso de CWC. En lo referente a logs, éstos ingresarán directamente a disco, pues esta información es de solo lectura y no será analizada en tiempo real para tomar decisiones.

El tiempo entre cada descarga será definido por el administrador del sitio web, hay que tomar en cuenta que debido a que esta información se va a mantener en memoria volátil, ésta se puede perder, pero este es un riesgo que se asumirá en favor de brindar un mejor servicio.

Debido a que el periodo de tiempo entre cada descarga es configurable, si se desea reducir el riesgo de perder información, este tiempo deberá tender a ser corto, de lo contrario, puede ser tan largo como se desee.

8.1.3 CWC Core

En la capa CWC Core es donde se implementa toda la funcionalidad de la Comunidad Web Caché, en esta capa se implementará:

- Administración de Objetos Cacheables.
- Administración de miembros de la CWC.
- Administración de estadísticas.
- Se contara con un módulo para toma de decisiones en base a las estadísticas recolectadas.
- Comunicación con el servidor web.
- Comunicación con los miembros de las caches.

La capa CWC Core implementará toda la funcionalidad fundamental de la Comunidad Web Caché, ésta tendrá interacción con:

- La capa CWC Data Management para escribir y leer datos.
- La capa CWC Web Interface para comunicarse con el servicio web.
- La capa CWC HTTP Interpreter para recibir las peticiones, analizarlas y determinar si éstas son válidas y utilizan el Protocolo CWC. Además enviará mensajes utilizando el este mismo protocolo.
- La capa CWC Cache Catalog para comunicarse con las caches, para recibir y enviar mensajes con estas, tendrá una conexión permanente con éstas para realizar la comunicación.

8.1.4 CWC Caches Catalog

Esta capa es la que se encarga de tener una conexión permanente con las caches y también de intercambiar cualquier mensaje con éstas, es el módulo que se comunica a través del protocolo CWC, está constantemente intercambiando mensajes con las cachés para recolec-

tar estadísticas, para indicar cambios de estados en los objetos, entre otros.

8.1.5 CWC Protocol Interpreter

En esta capa se realiza el análisis de cualquier petición utilizando el protocolo HTTP, se identifica si la petición utiliza correctamente el protocolo CWC, si es así se pasa el mensaje a la capa CWC Core para que sea atendida. Cuando su atención termine recibirá la respuesta y la enviará al cliente que realizó la petición, en el caso de que la petición no utilice el protocolo CWC será enviada al CWC Web Interface que actuará como un simple puente hacia el servidor web.

8.1.6 CWC Web Interface

Como se ha mencionado anteriormente, el CWC Server se implementará como un módulo de un servidor web existente.

Este módulo será una capa que actuará como un interfaz para intercambiar datos con las estructuras internas del servicio web y para reenviar las solicitudes que deban ser atendidas por éste.

8.2 CWC CLIENT

En la figura 14 se muestra el esquema general del mecanismo de comunicación del CWC Client. Este actuará como un "proxy web cache" para las peticiones que envía el cliente.

Por ejemplo, si se envía una petición a <http://www.cwc.com> esta petición será interceptada por el CWC Client y resuelta mediante el protocolo CWC, al final el CWC Client se encarga de obtener el archivo desde las diferentes cachés y ensamblarlo, además también se ocupa de servirlo al navegador web. El uso de una Comunidad Web Caché es totalmente transparente para el usuario y el navegador web. Una versión más detallada de lo que sucede es la siguiente:

1. El CWC Client, intercepta la petición.
2. Se verifica que el Servidor Web utilice el protocolo CWC.
3. En el caso de que no se utilice, se realizará la petición directamente al servidor.
4. En el caso de que se utilice, se solicitará la lista de cachés con el objeto al servidor.
5. Se inicia la descarga distribuida.
6. Se vuelve a ensamblar el archivo una vez que se tengan todas las partes.

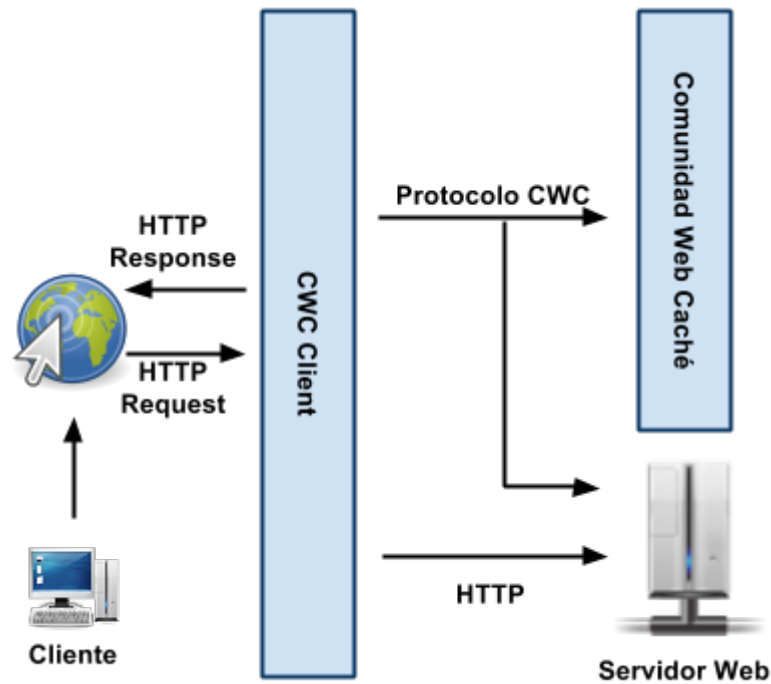


Figura 14: Comunicación del CWC Client

7. Se envía el resultado al navegador web.

La arquitectura del CWC Client es muy similar al de CWC Server. En la figura 15 se muestra una descripción gráfica, donde las principales diferencias radican en:

1. No cuenta con una base de datos.
2. No cuenta con un servidor web.
3. Funcionará como una extensión para el navegador web.

8.2.1 Data Access

En esta capa se implementará toda la lógica para el acceso a datos, en este caso solo se realizarán accesos a disco, por lo mencionado anteriormente, pues no cuenta con una base de datos del lado del cliente y además porque la principal funcionalidad de ésta es realizar la transmisión de archivos a disco.

8.2.2 CWC Data Management

Esta capa también como en el CWC Server, funciona como una cache de los datos, esto para reducir los accesos a disco que son sumamente costos. En caso que un archivo sea leído desde multiples localizacio-

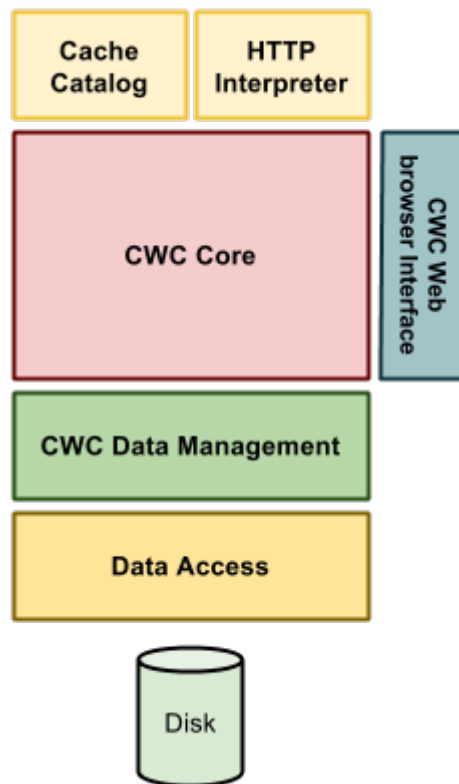


Figura 15: Arquitectura CWC Client

nes, éste acceso se agilizará a través un la memoria de acceso aleatoria (RAM).

8.2.3 CWC Client Core

La capa CWC Client Core, implementará toda la funcionalidad básica de la Comunidad Web Caché, como lo es:

1. Interacción con el navegador web o agente de usuario.
2. Administración de la cache.
3. Verificación de Consistencia.
4. Administración de la membresía a la comunidad.
5. Servir un archivo.

8.2.4 CWC Cache Catalog

Esta capa implementa el protocolo CWC en forma pura, cuando se trata de una caché, ésta capa se encarga de realizar toda la comunicación entre la cache y el servidor, mantiene una conexión con el servidor e intercambia mensajes.

8.2.5 CWC HTTP Interpreter

Esta capa implementa de manera muy básica el protocolo HTTP, sirve como cliente para hacer los HTTP Distributed Get y para servir partes de archivos. Además sus funciones son:

1. Realizar las peticiones HTTP.
2. Servir Archivos.

8.2.6 CWC Web Browser Interface

Esta se encarga de integrarse con la arquitectura del navegador web o el agente de usuario, en este caso se integra para capturar todas las peticiones HTTP y enviarlas en formato CWC.

DISEÑO DEL PROTOCOLO CWC

9.1 MENSAJES HTTP DEL PROTOCOLO CWC

9.1.1 *DGET*

Esta tal vez es una de los métodos de mayor importancia en el CWC, pues establece como obtener un archivo desde múltiples cachés. Este método se rige bajo el siguiente flujo:

1. Se verifica que el servidor web soporte el protocolo CWC.
2. Se solicita el top de miembros que tienen el archivo que se desea descargar. El número de miembros que se enviarán por petición se limitará a 100.
3. Se solicitan los atributos, Checksum y tamaño al servidor.
4. Si el tamaño del archivo es menor a 1MB, se solicita desde un solo miembro.
5. Si el tamaño del archivo es mayor a un MB, se divide en segmentos y se asigna de la siguiente manera:
 - a) Al primer miembro del top se le asigna un equivalente al 10 % del tamaño del archivo en partes, si un 10 % es menor a un 1MB, se le asigna el tamaño correspondiente a 1MB.
 - b) Al segundo se le asigna el equivalente a un 10 % del tamaño del archivo en partes, si un 10 % es menor a 1MB, se le asigna el tamaño correspondiente a 1MB.
 - c) Se realiza este procedimiento hasta llegar al quinto de la lista, después de esto se reparte equitativamente entre el resto de miembros.
 - d) Si se da el caso de que el número de miembros es menor a 10, se reparte el total del archivo en partes iguales entre todos los miembros.
6. Se inicia la descarga simultánea desde todos los miembros. En la figura 16 se puede observar este proceso.
7. En el caso de que alguno de los miembros falle, se recurrirá al siguiente de mayor rango (mejor posicionado en el top) si se diera el caso de que no se tienen más miembros a quien recurrir se solicitará el archivo o el segmento al servidor.

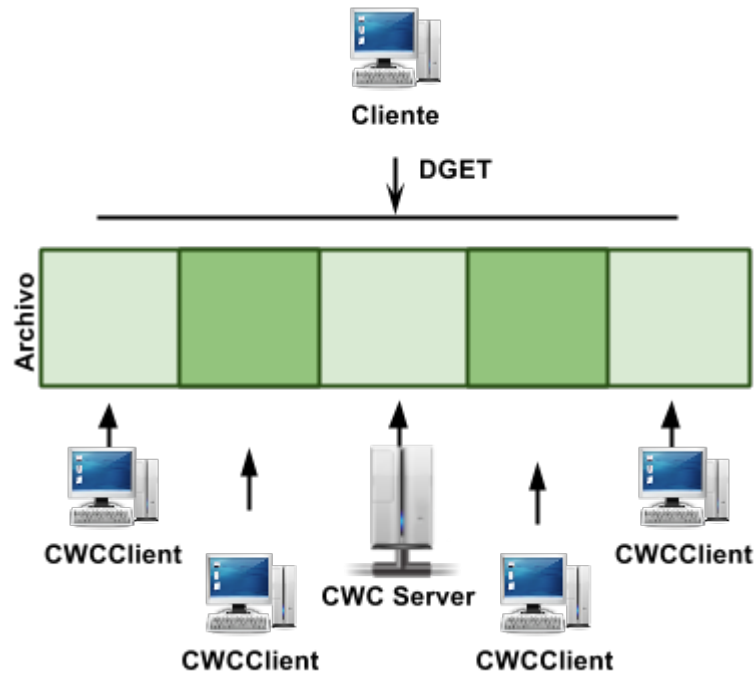


Figura 16: Funcionamiento del DGET

8. Cuando se ha terminado la descarga, se unen todas las partes y se envía el archivo al navegador web, como si el archivo fuera servido por un servidor web local.

El DGET recibirá los siguientes parámetros:

1. Id del archivo.
2. Offset en partes.
3. Tamaño que debe leer.

Por ejemplo, el archivo que se desea leer es el Id 10, y se debe leer a partir del byte 95, más los 50 bytes siguientes, entonces la llamada sería `DGET(10, 95, 50)`.

9.1.2 *DSEND*

Este es el caso contrario de DGET, es necesario recordar que un DGET tiene como parámetros el Id del archivo, el offset en partes y por último el tamaño que se debe leer en segmentos. Data esta información se puede enviar un flujo de segmentos hacia quien hizo el DGET.

9.2 ESTRUCTURA DE LOS MENSAJES CWC

El protocolo de comunicación se basará en XML para realizar el envío y recepción de mensajes entre el servidor, los clientes y las cachés. En el presente apartado se definirá los mensajes que se intercambiarán y sus respuestas.

Cualquier mensaje que se envíe siempre tendrá tres componentes, que servirán como identificador de éste:

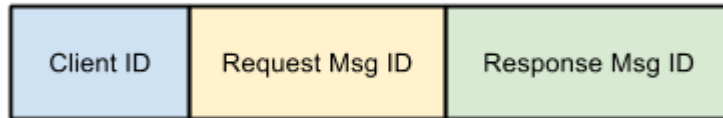


Figura 17: Estructura de un mensaje

En la figura 17 se observan los componentes del mensaje, los cuales son:

1. El identificador único del cliente en el sistema, este es generado cuando un cliente entra a formar parte de la comunidad, en el caso del servidor su identificador único es 0. En algunos mensajes se utilizará un identificador por defecto (-1) este significará que hay un nuevo cliente que desea convertirse en un miembro de la comunidad pero aun no tiene su identificador único.
2. Request Message Id: Este es el número del mensaje al que se esté respondiendo, cuando se encuentre el caso que no se esté respondiendo a ningún mensaje se introducirá un número negativo lo cual indicara que se está iniciando una comunicación.
3. Response Message Id: Este es el número del mensaje que estamos enviando, este número se genera como un aleatorio entre 1 y 10000, en algunos casos el mensaje no necesita respuesta entonces se introduce un número negativo.

9.3 OPERACIONES DEL PROTOCOLO CWC

En el presente apartado se definirán las operaciones necesarias para el funcionamiento del protocolo CWC. Aquí se detallará tanto valores de entrada, como de salida y además la estructura esperada.

9.3.1 *ACK*

Este mensaje es una confirmación. Además puede ser enviado por cualquiera de las partes involucradas en la comunicación: Servidor o Cliente. Éste tiene la siguiente estructura:

```
<ack clientid="X" requestid="X" responseid="-1"> </ack>
```

Listado 7: Mensaje ACK

RESPUESTA No necesita respuesta.

9.3.2 *GetClientId*

Este mensaje se utiliza cuando un nuevo CWCCient quiere formar parte de la comunidad. La estructura del mensaje es:

```
<getClientId clientid="-1" requestid="-1" responseid="X">
</getClientId>
```

Listado 8: Mensaje GetClientId

Como se puede observar el `clientId` es -1 ya que es un cliente nuevo, el `requestId` es -1 ya que no se está respondiendo a nadie y el `responseid` es un numero aleatorio entre 1 y 10000.

RESPUESTA La Respuesta esperada por parte del servidor es:

```
<getClientId clientid="0" requestid="X" responseid="Y">
  <client> Z </client>
</getClientId>
```

Listado 9: Mensaje de Respuesta GetClientId

Para terminar la comunicación se espera por parte del cliente un ACK con su nuevo `clientId`.

9.3.3 *UpdateMembership*

Este mensaje se utiliza para establecer el tipo de membresía, recursos que se están prestando a la caché y tipo de administración. La primera vez que se utiliza, lo que se busca es crear la membresía, lo usos subsecuentes son básicamente para actualizar la información. La estructura del mensaje debería ser:

```
<UpdateMembership clientid="X" requestid="-1" \
  responseid="Y">
  <type> Z </type>
  <manage> Z </manage>
```



```

    <disk> Z </disk>
    <mem> Z </mem>
    <upload> Z </upload>
</UpdateMembership>

```

Listado 10: Mensaje de UpdateMembership

Los valores que pueden tomar estas variables son:

- **type:** 1 = Voluntaria, 2 = Involuntaria
- **manage:** 1 = Atendida, 2 = desatendida, 3 = Administrada por servidor
- **disk:** En MB un número mayor o igual a 100
- **mem:** En MB un número mayor o igual a 100
- **upload:** En KB un número mayor o igual 128

RESPUESTA Se espera como respuesta un ACK por parte del servidor.

9.3.4 *GetObjectInfo*

Este mensaje se utiliza para obtener la información de un objeto cacheable. La estructura es:

```

<GetObjectInfo clientid="X" requestid="-1" responseid="Y">
    <objectId> Z </objectId>
    <objectLocation> Z </objectLocation>
</GetObjectInfo>

```

Listado 11: Mensaje de GetObjectInfo

Los valores que pueden tomar estas variables son:

- **objectId:** El identificador único del objeto.
- **objectLocation:** El URI relativo del objeto cacheable.

Es posible que se incluya solo uno o ambos.

RESPUESTA La Respuesta esperada por parte del servidor es:

```

<response clientid="0" requestid="X" responseid="Y">
    <checksum> Z </checksum>
    <id> Z </id>
    <location> Z </location>
    <size> Z </size>
    <expdate> Z </expdate>
</response>

```

Listado 12: Mensaje de Respuesta de GetObjectInfo

Lo valores que retorna son:

- **checksum:** El Checksum del archivo
- **id:** El identificador del objeto
- **location:** URI relativo donde se encuentra el objeto.
- **size:** Tamaño del archivo en megabytes
- **expdate:** Fecha de expiración.

La respuesta esperada por parte del cliente es un ACK.

9.3.5 *ModifyObject*

Este mensaje es enviado por el servidor, en este caso se ha realizado un cambio del lado del servidor que debe ser propagado hacia los clientes. La estructura del mensaje es:

```
<ModifyObject clientid="0" requestid="X" responseid="Y">
  <checksum> Z </checksum>
  <id> Z </id>
  <location> Z </location>
  <size> Z </size>
  <expdate> Z </expdate>
  <type> Z </type>
</ModifyObject>
```

Listado 13: Mensaje de ModifyObject

La única diferencia con el anterior es el type, el cual especifica el tipo de modificación que se está realizando, 1 significa una modificación soft la cual no necesita actualizar el archivo y cualquier otra cosa es una modificación hard lo que implica que hay que deshabilitar el archivo y volverlo a cargar.

RESPUESTA La respuesta esperada por parte del cliente es un ACK.

9.3.6 *Enable-Disable Object*

En este caso el mensaje se utiliza para habilitar o deshabilitar un objeto que se encuentra en la caché. Este mensaje puede ser enviado tanto por el servidor como por el cliente. La estructura del mensaje es:

```
<EnableObject clientid="X" requestid="-1" responseid="Y">
  <objectId> Z </objectId>
</EnableObject>
```

Listado 14: Mensaje de EnableObject

Se envía el identificador del objeto, si el objeto esta Enable se cambia a Disable o viceversa y se espera un ACK de parte de la otra parte.

RESPUESTA La respuesta esperada por parte del cliente o servidor, según corresponda, es un ACK.

9.3.7 DeleteObject

Este mensaje es utilizado ya sea por los clientes o por el servidor, cuando un objeto se borra en alguno de los dos. La estructura del mensaje es:

```
<DeleteObject clientid="X" requestid="-1" responseid="Y">
  <objectId> Z </objectId>
</DeleteObject>
```

Listado 15: Mensaje de DeleteObject

Se envía el identificador del objeto y se registra el borrado, en el caso del cliente borra el objeto de la cache, en el caso del servidor borra la relación entre la cache y el objeto.

RESPUESTA La respuesta esperada por parte del cliente o servidor, según corresponda, es un ACK.

9.3.8 GetMemberStatus

Este mensaje es utilizado para solicitar el estado de un miembro de la caché (no del servidor), se puede ejecutar tanto por el servidor como por cualquier miembro de la caché. La estructura del mensaje es:

```
<GetMemberStatus clientid="X" requestid="-1" responseid="Y">
  <memberId> Z </memberId>
</GetMemberStatus>
```

Listado 16: Mensaje de GetMemberStatus

RESPUESTA Se envía el memberId que es igual al clientId, se espera la siguiente respuesta:

```
<response clientid="0" requestid="X" responseid="Y">
  <status> Z </status>
</response>
```

Listado 17: Mensaje de Respuesta de GetMemberStatus

El status puede tomar los siguientes valores 0 = desconocido, 1 = activo, 2 = no-activo, 3 = offline y 4 = online.

Se espera un ACK por parte de quien solicitó el status.

9.3.9 *ChangeMemberStatus*

Este mensaje es utilizado para cambiar el status de un miembro de la cache, este solo puede ser ejecutado por parte del servidor hacia cualquiera de los clientes o por parte del cliente hacia el servidor para cambiar su estatus. La estructura del mensaje es:

```
<ChangeMemberStatus clientid="0" requestid="X" \\  
    responseid="Y">  
    <memberId> Z </memberId>  
    <status> Z </status>  
</ChangeMemberStatus>
```

Listado 18: Mensaje de ChangeMemberStatus

El status puede tomar cualquier valor de la operación definida en 9.3.8.

RESPUESTA La respuesta esperada por parte del cliente es un ACK.

9.3.10 *GetListOfFiles*

Este mensaje se utiliza ya sea para pedir una lista de archivos a un cliente con su respectivo checksum o para solicitar una lista de archivo que debería tener un cliente en su caché con su respectivo checksum. La estructura de este mensaje es:

```
<GetListOfFiles clientid="0" requestid="X" responseid="Y">  
    <clientId> Z </clientId>  
</GetListOfFiles>
```

Listado 19: Mensaje de GetListOfFiles

RESPUESTA La Respuesta esperada por parte del cliente es:

```
<response clientid="0" requestid="X" responseid="Y">  
    <file objectId="Z" checksum="Z"> </file>  
    <file objectId="Z" checksum="Z"> </file>  
    <file objectId="Z" checksum="Z"> </file>  
</response>
```

Listado 20: Mensaje de Respuesta de GetListOfFiles

Se espera un ACK por parte de quien recibe la lista de archivos.

9.3.11 *GetStats*

Se ejecuta por parte del servidor, para obtener las estadísticas de una de las cachés, la estructura del mensaje es:

```
<GetStats clientid="X" requestid="-1" responseid="Y">
  <clientId> Z </clientId>
</GetStats>
```

Listado 21: Mensaje de GetStats

RESPUESTA La Respuesta esperada por parte del cliente es:

```
<response clientid="0" requestid="X" responseid="Y">
  <maxupload> Z </maxupload>
  <minupload> Z </minupload>
  <maxclients> Z </maxclients>
  <currentclients> Z </currentclients>
</response>
```

Listado 22: Mensaje de Respuesta de GetStats

Se envía la máxima velocidad de transferencia, la mínima velocidad de transferencia, máxima cantidad de clientes simultaneos y clientes actuales.

Se espera un ACK por parte de quien recibe las estadísticas.

9.3.12 *SendStats*

En algunos casos las estadísticas son enviadas por el mismo cliente, en este caso la estructura del mensaje es:

```
<SendStats clientid="0" requestid="X" responseid="Y">
  <maxupload> Z </maxupload>
  <minupload> Z </minupload>
  <maxclients> Z </maxclients>
  <currentclients> Z </currentclients>
</SendStats>
```

Listado 23: Mensaje de SendStats

RESPUESTA La Respuesta esperada por parte del servidor es un ACK.

9.3.13 *GetTopCachesbyObject*

Este mensaje es enviado por los clientes, para solicitar el top de cachés que pueden servir un objeto. El formato del mensaje es:

```
<GetTopCachesbyObject clientid="0" requestid="X" \\
  responseid="Y">
  <objectId> Z </objectId>
</GetTopCachesbyObject>
```

Listado 24: Mensaje de GetTopCachesbyObject

RESPUESTA La Respuesta esperada por parte del servidor es:

```
<response clientid="0" requestid="X" responseid="Y">
  <cache clientId="Z" URL="Z"> </cache>
  <cache clientId="Z" URL="Z"> </cache>
  <cache clientId="Z" URL="Z"> </cache>
</response>
```

Listado 25: Mensaje de Respuesta de GetTopCachesbyObject

Se espera un ACK por parte del cliente que recibió esta lista.

9.3.14 GetConfParams

Este también es únicamente ejecutado por el cliente, se utiliza para tomar la configuración del sistema, hay varios parámetros que pueden cambiar, este mensaje se utiliza para obtener estos cambios. La estructura del mensaje es:

```
<GetConfParams clientid="0" requestid="X" responseid="Y">
</GetConfParams>
```

Listado 26: Mensaje de GetConfParams

RESPUESTA La Respuesta esperada por parte del servidor es:

```
<response clientid="0" requestid="X" responseid="Y">
  <param paramId="Z"> </param>
  <param paramId="Z"> </param>
  <param paramId="Z"> </param>
</response>
```

Listado 27: Mensaje de Respuesta de GetConfParams

Se espera un ACK por parte de quien recibe la lista de parámetros.

9.3.15 IsAlive

Este mensaje se utiliza para saber si algún miembro del sistema está vivo. La estructura del mensaje es:

```
<IsAlive clientid="X" requestid="-1" responseid="Y">
</IsAlive>
```

Listado 28: Mensaje de IsAlive

RESPUESTA Para este mensaje se espera un ACK por parte de la cache representada por `clientId`.

9.4 SEGUIMIENTO DE LOS MIEMBROS CONECTADOS

Es de suma importancia saber si alguno de los miembros de la CWC se encuentra bien, para esto se deberá estar revisando que los miembros estén en línea, para lograrlo, se seguirá:

1. El servidor tendrá una estampa de tiempo para cada cache, la cual indicara la última vez que se recibió un mensaje por parte de esta.
2. Cada cache tendrá una estampa de tiempo del servidor que indicara la última vez que este envió un mensaje.
3. Existirá un parámetro configurable llamado "tiempo sin respuesta", el cual establece cuanto tiempo debe pasar después de que se recibió un mensaje para preocuparse.
4. Cada vez que se recibe un mensaje por parte de alguna caché o el servidor, se establece esta estampa a la hora en la cual se recibió el mensaje.
5. Se estará revisando periódicamente si alguna estampa ya es lo suficientemente antigua como para que el tiempo actual menos la estampa del último mensaje se mayor o igual al "tiempo sin respuesta".
6. Si ha pasado más tiempo del establecido por "tiempo sin respuesta", se envía un mensaje de `IsAlive`.
7. Si no se recibe respuesta después de tres intentos se establece que esa caché o servidor esta caído.
8. En el caso de una cache, se seguirá revisando periódicamente en tiempos no menores a "tiempo sin respuesta" hasta que se descubra que está arriba.
9. En el caso del servidor se actualizará la caché como offline y se esperará a que ésta se comunique.

9.5 RECUPERACIÓN DE FALLAS

Cuando se presenta una falla, se puede dar ya sea en el lado del servidor como del lado de alguno de los clientes, el protocolo que se seguirá es:

9.5.1 ¿Cómo descubrir una falla?

Para descubrir una falla se deben de dar las siguientes condiciones:

1. En el caso del servidor perder contacto con todas las caches y con Internet (una implica a la otra).
2. En el caso de un cliente perder la conexión con el servidor o con Internet (una implica a la otra).

9.5.2 Falla del Servidor

Una falla se puede dar por muchas razones, no es nuestro objetivo estudiarlas, el caso es que se pueden dar de dos tipos:

- Se pierda información volátil: Esto quiere decir que el servidor por ejemplo se apague, lo cual hace que la información que estaba en memoria y no se había enviado a disco o a base de datos mediante un flush, se perdió.
- Que no se pierda información volátil: Por ejemplo que se pierda la conexión a Internet o que se apague voluntariamente el servidor entonces se cortan las funciones de red.

En cualquier caso este es el protocolo:

1. Si es sin pérdida de información volátil: Se debe hacer un flush inmediatamente.
2. Una vez que nos hallamos recuperado de la falla, en este caso nos damos cuenta de que fue así ya que tenemos comunicación con las caches y con Internet, dejamos a todas las cache en un estado desactivadas.
3. Cargamos la información a memoria.
4. Esperamos la petición por estado, cuando el estado es desactivado se procede a revisar la consistencia.
5. Esperamos que recibir los mensajes por parte de las caches, que ante esta eventualidad, enviaran mensajes GetListOfFiles.
6. Esperamos hasta que las caches envíen un cambio (ChangeStatus) de estado a en línea.
7. Esperamos que se repita esto las veces que sean necesarias.
8. Después de que las caches estén en línea, retomaran los mensajes donde habían quedado (recordemos se guarda el último mensaje enviado, si no se recibe respuesta se declara que está abajo el servidor hasta que responda y se retoma la comunicación en el último mensaje enviado)

9.5.3 *Falla un Cliente*

En este caso la cache por alguna razón perdió comunicación y el servidor la marco como inactiva, cuando se recupera la comunicación lo primero que se solicita es el estado de la cache, este recibirá un desactivada entonces pedirá los archivos para revisar la consistencia, cuando todo esté correcto solicitara volver a estar en estado en línea y seguirá la comunicación como estaba anteriormente. Para reconstruir el DGET, el cliente deberá evaluar la cantidad de partes descargadas y solicitar la lista de caches disponibles al servidor para retomar la descarga.

9.6 IMPLEMENTACIÓN DEL MODELO DE CONSISTENCIA

Como se mencionó anteriormente el modelo de consistencia no es tan complejo debido a que solo un actor es el que modifica los objetos que se incluyen en la caché, este actor es el CWCServer, el modelo de consistencia bastará con:

- Si se modifica un objeto en el servidor, este ejecutara un ModifyObject hacia todas las caches que tienen copias de este objeto, si es un cambio de archivo tamaño, entre otros. Decidirá si es necesario eliminar el archivo y volver a solicitarlo.
- Las caches estarán revisando periódicamente, la consistencia, esto quiere decir que solicitara al servidor el Checksum de los archivos que tiene en su cache, lo calculara y lo comparara.
- Si hay algo extraño, simplemente solicita que se envíe el archivo de nuevo, si todo está bien pues no hay ningún problema.
- Después de una falla se revisa la consistencia, esto aplica para todas las caches en el caso de que sea una falla de servidor o para la cache que fallo.

DISEÑO DEL CWC

CWC es un modelo de web caching distribuido, el cual busca hacer un uso eficiente de los recursos computacionales disponibles tanto en el servidor como en los clientes mediante el establecimiento de una comunidad. Esta comunidad tiene como objetivo que los clientes proporcionen una cantidad finita de recursos para establecer una Web Cache Distribuida la cual permita reducir el tráfico sobre el servidor, ya que los archivos serán servidos desde múltiples localizaciones y en una mayoría de casos, sin trabajo adicional por parte del servidor, más que indicar donde se encuentra el archivo y como accederlo.

Con el establecimiento de la Web Cache Distribuida y al reducir el tráfico sobre el servidor, se permitirá atender a más clientes y hacer la comunidad mucho más grande. Lo cual, en última instancia, implica una Web Cache Distribuida mucho más grande y con más recursos.

10.1 CWC SERVER

El CWC Server es una aplicación que se Integra con el servidor web, su función consiste en realizar la administración de la CWC e interceptar las peticiones de los clientes que utilizan CWC. Entre las funciones del CWC Server se encuentran:

- Administración del Caché local
- Administración del CWC
- Estadísticas
- Segmentación de Archivos
- Administración Remota de Caché

10.1.1 *Administración de Objetos en la Caché local*

En un servidor web existen gran cantidad de objetos, que van desde los documentos HTML y CSS más comunes hasta lo que hoy conocemos como multimedia, que incluye un gran número de archivos de gran tamaño como video y música. Además también existen de aplicaciones de tiempo real como video conferencias y streaming de video y audio. La administración de Objetos en la caché local se refiere a como identificar cuales objetos se pueden agregar a la caché del CWC y cuales no. Cada objeto contara con los siguientes atributos:

- *Checksum*: Esto con el objetivo de saber si el archivo esta corrupto. No nos preocuparemos por el cálculo del Checksum.
- *Id*: El archivo tendrá su identificador único, generado automáticamente.
- *Localización*: URI relativo al URL del sitio web.
- *Cachés*: Cuáles clientes CWC poseen una copia de este objeto.
- *En Caché*: Bandera que indica si el objeto puede ser compartirse a través del CWC o no.
- *Tiempo expiración*: Dentro de cuanto el objeto será obsoleto.

Todos estos atributos, excepto uno, deben de perdurar en disco, para evitar fallas eléctricas o bien el apagado del equipo. El atributo *Cachés* será mantenido en memoria, pues es un dato que se calcula en tiempo de ejecución de la CWC.

A continuación se presentan las acciones presentes en la Administración de Objetos en la Caché local:

AGREGAR UN NUEVO OBJETO Cuando se agrega un nuevo objeto al servidor web se deben calcular sus atributos y almacenarlos. Cuando una nueva cache obtiene un objeto se actualiza su información de caches, lo cual quiere decir que este objeto puede ser servido desde una cache más.

MODIFICAR UN OBJETO Existen dos tipos de modificaciones:

SOFT Una modificación soft no implica que la cache tenga que volver a jalar el objeto, una modificación soft es cambiar la localización del archivo, cambiar las caches del archivo, cambiar el estado de Cacheable, cambiar su tiempo de expiración.

HARD La modificación hard, se refiere al cambio del tamaño del archivo lo cual implica un cambio de su Checksum.

Cada atributo implica diferentes que deben ser tomadas, entre estas se tiene:

- *Checksum y Tamaño*: Al darse un cambio en estos atributos se deben invalidar todas las caches de este objeto y notificar a las mismas para que borren sus copias y si es necesario que obtengan una copia fresca.
- *Id*: No se hacen cambios de Id.
- *Localización*: tiempo de expiración y Caches: Simplemente se actualiza la información no se notifica a las caches.
- *En Caché*: Se notifica a las caches el nuevo estado para que invaliden y borren su copia.

BORRAR UN OBJETO Cuando se borra un objeto, se borra el objeto junto con sus atributos y además se notifica a las caches para que borren su copia.

VERIFICAR EXPIRACIÓN El CWC Server deberá estar revisando si los objetos de la cache se encuentran válidos, en el caso de que alguno sea inválido se establece el objeto como no-cacheable y se trata como una modificación.

EN TRANSFERENCIA Si se da una modificación hard o un borrado mientras el objeto se está transfiriendo, como política se deben invalidar todas las copias incluyendo la original y borrarlas hasta que se termine de servir a todos los clientes que están recibiendo el archivo.

10.1.2 Administración del CWC

Esta administración es muy relativa ya que no se tendrá control completo sobre ésta, en este caso nos referimos a los clientes que conforman la comunidad y a los objetos que pueden ser compartidos en la caché.

Cuando el CWC Server se inicia por primera vez, posee una caché que vamos a definir como *fría*, lo cual quiere decir que no existen miembros en la comunidad y por ende no existen objetos cacheados. Las acciones que se pueden dar son las siguientes:

AGREGAR UN NUEVO MIEMBRO Cuando se agrega un miembro se crea en memoria la estructura para este cliente y se establecen los recursos aportados por este cliente, dentro de la información importante que se debe almacenar, se encuentra:

- Los clientes que está sirviendo.
- Los archivos que tiene en su caché.
- La localización geográfica
- El estado del cache: Activa/desactiva.
- El estado del cliente: online/offline.

AGREGAR UN MIEMBRO EXISTENTE Se puede haber dado el caso que el servidor fallara, por lo que la información de las caches se perdería. O bien, que alguna de las caches fallara y se perdiera la conexión con el servidor. Esto último implicaría que la cache quedara en un estado offline, o se desactivara. Para agregar una cache existente sería necesario:

1. La cache se comunica con el servidor, ya sea porque se recupero de alguna falla o se activo de nuevo o porque el servidor fue el que fallo y ya está arriba de nuevo.

2. El servidor busca en su lista para ver si la cache se encuentra disponible.
3. En el caso de que se encuentre:
 - a) Se le envía la lista de archivo que debería tener cacheados y su respectivo Checksum.
 - b) La cache verifica que tiene los archivos y que son correctos.
 - c) Si todo está bien la cache confirma para ser promovida a online.
 - d) Si algún archivo esta corrupto o no existe, se borra y se envía la invalidación al servidor.
 - e) Se confirma el estado consistente para ser promovida a online.
4. En el caso de que no se encuentre:
 - a) Se le solicita una lista de archivos cacheados y su Checksum.
 - b) Cuando la cache los envía, se verifican los archivos y su Checksum.
 - c) Si todo está bien se confirma su estado online al crear las estructuras del miembro con sus atributos.
 - d) Si alguna archivo fue borrado, modifica o esta corrupto, se indica que se invalida y una vez que se confirma la invalidación se crean las estructuras del miembro y se le notifica a la cache que esta online.

MODIFICAR UN MIEMBRO Cualquiera de los atributos de un miembro se puede modificar, lo más importante es conocer si éste está activo/no activo, esto se logra mediante la revisión periódica del estado de las cachés.

10.1.3 Estadísticas

En una comunidad al igual que en cualquiera, hay diversidad de miembros, algunos cuentan con muchos recursos, otros con pocos o algunos fallan bajo presión. La idea con las estadísticas, es que una vez que se solicita un objeto, enviar la lista de caches que lo tienen disponible, ordenada de tal manera que se presenten los que históricamente tiene mejores referencias de primero.

Esto no quiere decir que a los que tienen un bajo historial se dejarán fuera, si no que cuando se deba atender una petición tendrán menor trabajo que el resto. Tampoco quiere decir que se quedarán haciendo el peor trabajo siempre, sino más bien, se pretende implementar un

sistema de puntos en el cual, presente la oportunidad de crecer en confianza y convertirse en una de las caches más altas en el ranking.

Entre las estadísticas que se van a recolectar son:

LOGS DE ACCESO Se debe llevar un recuento para cada petición, de quien la atendió y tiempo de respuesta.

BOOKKIPPING Para cada una de las caches se debe tener un histórico de:

- Tiempo de respuesta.
- Número de clientes atendidos por unidad de tiempo.
- Número de veces que ha fallado.
- Velocidades de transferencia mínimas y máximas.
- Archivos en cache y número de veces que han sido transmitidos.

SISTEMA DE PUNTOS El sistema de puntos funcionará de la siguiente manera:

PUNTOS ESTÁTICOS Estos puntos son dados en el transcurso normal de la vida del CWC.

- La máxima cantidad de puntos es 100 y la mínima es -100.
- Un miembro nuevo, recibirá una bonificación de 50 puntos por formar parte de CWC.
- Cada 6 horas que el miembro permanezca online, recibirá una bonificación de 1 punto con un máximo de 4 puntos diarios. En el caso contrario perderá un punto cada 6 horas con un máximo de 4 diarios.
- Si el miembro se desactivo, perderá un punto cada 12 horas por estar desactivado. Con un máximo de 2 puntos diarios.
- Por cada cliente atendido satisfactoriamente el miembro recibirá un total de 2 puntos, por cliente atendido insatisfactoriamente el miembro perderá 3 puntos.

BONIFICACIONES Las bonificaciones son dadas por acciones extras beneficiosas hacia los clientes.

- El miembro de la cache con el mejor tiempo de respuesta ganara una bonificación de 10 puntos.
- Los miembros de la cache con tiempo de respuesta mayor a la media ganaran un punto adicional.
- El miembro de la cache que haya transmitido el mayor número de veces el objeto solicitado, recibirá un bono de 10 puntos.

CÁLCULO TOTAL A continuación se detallan las reglas que dictan como se realizará el cálculo total de los puntos acumulados:

1. Con las reglas y bonificaciones anteriormente mencionadas, se calcula el top de miembros de la cache.
2. Con el top de miembros listo, se eliminan todos aquellos que están atendiendo a su máximo número de clientes permitido que se calcula mediante:

$$\frac{\text{conexión de subida}}{\text{mínima velocidad para servir archivos}}$$

Esto quiere decir que si se asigna una velocidad mínima para servir archivos de 64Kbps y el miembro de la cache estableció su conexión de subida en 512Kbps, el número máximo de clientes que puede atender simultáneamente es 8.

3. Para evitar la sobrecarga de trabajo sobre una cache, las caches recibirán una penalización en base al número de usuarios que están atendiendo, la regla establece que una cache perderá:

$$\text{número de clientes actuales} * 3\text{puntos}$$

10.1.4 *Segmentación de Archivos*

Cuando se va a servir un archivo desde la cache, ésta se debe descomponer en un número de segmentos. Un posible tamaño de un segmento podría ser 128Kb y éste podría corresponder al ancho de banda mínimo de la CWC.

Por ejemplo, si se desea descargar un archivo de 10Mb lo se debe de descomponer en partes de 128Kb y esta será la unidad de descarga de la caché. El segmento debe ser establecido por el administrador del sitio web, entre más pequeña mejor.

10.1.5 *Administración Remota de Caché*

Como se mencionó anteriormente, es posible que el cliente delegue la administración de la caché local al servidor. En el caso, el servidor se encargará de decir cuando se debe tener un archivo en la caché y cuando no se debe. La decisión de retirar un archivo o agregar otro a la caché, toma en cuenta si los archivos están siendo solicitados frecuentemente. Entonces el servidor envía los siguientes mensajes a las cachés:

- *Agregar Objeto*: Envía el id archivo y la cache se encarga de solicitarlo.
- *Borrar Objeto*: Envía el id del archivo y la cache se encarga de borrarlo.

10.2 CWC CLIENT

El CWC Client es una aplicación que se integra con el navegador web o agente de usuario, la principal función de este es poder obtener archivos de la CWC. Entre las funciones del CWC Client se encuentran:

- Asociación a la comunidad
- Administración de la Caché local
- Otras funciones de administración

10.2.1 Asociación a la comunidad

Existen diferentes dos formas para formar parte de la comunidad:

CONTRIBUYENTE Funciona de la siguiente manera: el usuario instala la aplicación CWC Client e inicia la navegación por diferentes páginas. Cuando se encuentra un servidor que soporta el protocolo CWC y se solicita un objeto que se pueda compartir en la caché de CWC, el cliente empieza a formar parte de la comunidad (si tiene instalado CWC Client es porque se desean compartir los recursos) con los parámetros mínimos, se podría decir que obtiene una membresía mínima.

ACTIVA En este caso, el usuario quiere ser miembro de la CWC de un sitio en específico. Por lo que el usuario crea de manera específica la membresía y establece los recursos que desea compartir.

La diferencia radica en que si el usuario se convierte en miembro de manera voluntaria, la administración de la cache será realizada por el servidor. En todo caso, si la membresía es activa se puede cambiar a contribuyente.

Se puede dar el caso de que no desee formar parte de la comunidad de algún sitio en específico, entonces se puede configurar CWC Client de manera que ignore los sitios de una lista negra.

Para dejar de ser un miembro de la caché, se debe:

- Indicar que no se quiere pertenecer más a esta (Agregarla a la lista negra).
- No poseer ningún archivo de en caché (involuntaria).

10.2.2 *Administración de la Caché Local*

La administración de la cache se puede dar de 3 formas, atendida por el cliente, desatendida (por defecto cuando la membresía se es contribuyente) y por último administrada por el servidor (solo se utiliza cuando la membresía es activa):

DESATENDIDA En este caso, cuando la cache está llena, la misma cache escoge el archivo menos solicitado y lo borra, así sucesivamente hasta que se libera suficiente espacio como para almacenar el archivo.

ATENDIDA El usuario puede borrar archivos de la caché a su gusto. Para ello tendrá una interfaz de selección de archivos que se encuentren en la caché.

MANEJADA POR EL SERVIDOR El servidor por si solo decidirá cuales archivos deben estar en cache y cuáles no, sus decisiones serán transmitidas a la cache mediante el envío de mensajes.

10.2.3 *Otras Funciones de Administración*

Otras funciones de administración que posee el CWC Client son las siguientes:

INVALIDAR CONTENIDO Si se recibe un mensaje de que el contenido esta inválido, automáticamente se borra el objeto y se descarga.

REVISIÓN DE CONSISTENCIA Periódicamente se verifica si los contenidos de la cache son correctos, se le solicita al servidor el Checksum para los archivos que se encuentran en la cache y se comparan. Si están bien perfecto si no se procede a hacer una invalidación de contenido.

EL SERVIDOR ESTÁ VIVO Revisión constante del servidor, se debe establecer si se encuentra funcional o está fuera de línea.

MENSAJE DE OFFLINE, ACTIVO O DESACTIVO En cualquiera de estos casos la caché se comunica con el servidor para indicarle su nuevo estado.

TRANSFERENCIA DE ESTADÍSTICAS Periódicamente el cliente envía las estadísticas que se han producido con respecto a los objetos que se encuentran en su caché.

AGREGAR CONTENIDO A LA CACHÉ Es una acción ejecutada en el caso que se envíe la petición por parte del servidor o porque se ingrese a algún servidor web que utiliza CWC.

Parte IV

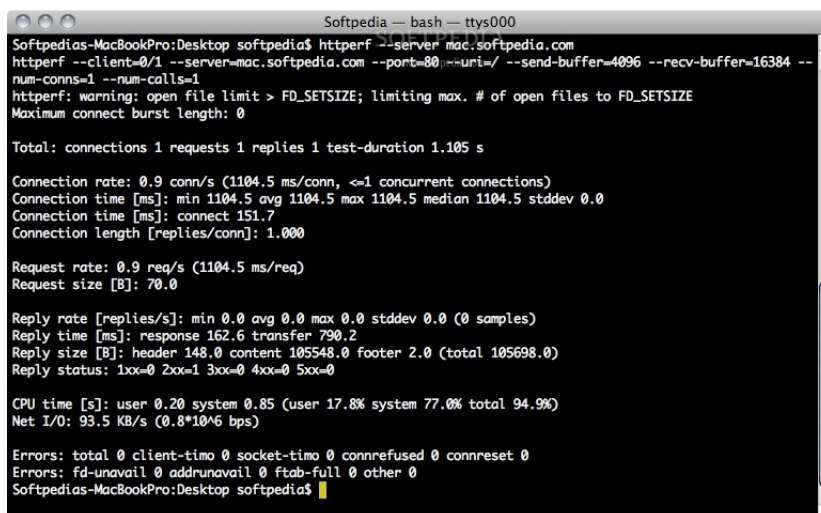
ANÁLISIS Y RESULTADOS

RENDIMIENTO

11.1 HERRAMIENTAS

11.1.1 *httperf*

Httpperf es una herramienta para medir el rendimiento de aplicaciones web. Provee una manera flexible de generar múltiples cargas de trabajo utilizando HTTP y con ellas poder medir el rendimiento Web. Puede observarse el funcionamiento de esta herramienta en la figura 18.



```

Softpedia — bash — ttys000
Softpedias-MacBookPro:Desktop softpedia$ httperf --server mac.softpedia.com
httperf --client=0/1 --server=mac.softpedia.com --port=80 --uri=/ --send-buffer=4096 --recv-buffer=16384 --
num-conns=1 --num-calls=1
httperf: warning: open file limit > FD_SETSIZE; limiting max. # of open files to FD_SETSIZE
Maximum connect burst length: 0

Total: connections 1 requests 1 replies 1 test-duration 1.105 s

Connection rate: 0.9 conn/s (1104.5 ms/conn, <=1 concurrent connections)
Connection time [ms]: min 1104.5 avg 1104.5 max 1104.5 median 1104.5 stddev 0.0
Connection time [ms]: connect 151.7
Connection length [replies/conn]: 1.000

Request rate: 0.9 req/s (1104.5 ms/req)
Request size [B]: 70.0

Reply rate [replies/s]: min 0.0 avg 0.0 max 0.0 stddev 0.0 (0 samples)
Reply time [ms]: response 162.6 transfer 790.2
Reply size [B]: header 148.0 content 105548.0 footer 2.0 (total 105698.0)
Reply status: 1xx=0 2xx=1 3xx=0 4xx=0 5xx=0

CPU time [s]: user 0.20 system 0.85 (user 17.8% system 77.0% total 94.9%)
Net I/O: 93.5 KB/s (0.8*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
Softpedias-MacBookPro:Desktop softpedia$
  
```

Figura 18: Funcionamiento de httperf

11.1.2 *OpenWebLoad*

OpenWebLoad es otra herramienta muy sencilla pero poderosa, es completamente en consola y permite hacer peticiones de archivos y entre la información que arroja esta número de ejecuciones, tiempo de respuesta de cada ejecución y tiempo de ejecución promedio. Puede observarse el funcionamiento de esta herramienta en la figura 19.

```

C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\sbr_admin\My Documents\Downloads>openload.exe forum.sbrforum.com 500
URL: http://forum.sbrforum.com:80/
Clients: 500
MaIps 0.77, Ips 0.77, Resp Time 1.239, Err 0%, Count 1
MaIps 0.77, Ips 0.79, Resp Time 2.504, Err 0%, Count 2
MaIps 1.06, Ips 3.71, Resp Time 3.092, Err 0%, Count 6
MaIps 1.41, Ips 4.57, Resp Time 3.554, Err 0%, Count 11
MaIps 1.90, Ips 6.26, Resp Time 4.756, Err 0%, Count 18
MaIps 2.00, Ips 2.94, Resp Time 6.399, Err 0%, Count 21
MaIps 3.46, Ips 16.57, Resp Time 7.585, Err 88%, Count 39
MaIps 5.61, Ips 25.00, Resp Time 8.898, Err 96%, Count 64
MaIps 6.44, Ips 13.86, Resp Time 9.021, Err 100%, Count 155
Total TPS: 9.11
Avg. Response time: 8.170 sec.
Max Response time: 9.154 sec.
Total Requests: 155
Total Errors: 131
C:\Documents and Settings\sbr_admin\My Documents\Downloads>

```

Figura 19: Funcionamiento de OpenWebLoad

11.1.3 Time

El comando `time` puede servir para obtener el tiempo que tarda una aplicación en completarse. En este caso se puede combinar con cualquier otra aplicación, por ejemplo con el prototipo del CWC. Puede observarse el funcionamiento de esta herramienta en la figura 20.

```

[personal] user@personal: ~/kmoragas@gmail.com/TEC/Maestria/Tesis/doc-http_p2p-es/code
File Edit View Search Terminal Help
[user@personal code]$ time python cwcserver.py

real    0m2.064s
user    0m0.042s
sys     0m0.022s
[user@personal code]$

```

Figura 20: Funcionamiento de time

11.1.4 JMeter

JMeter forma del proyecto TomCat y es una herramienta muy utilizada para realizar pruebas de carga en un sitio web, mediante esta herramienta se puede describir prácticamente todo el comportamiento de un usuario dentro de un sitio web, desde cuando se autentica hasta cuando se llenan campos de texto en un formulario. Puede observarse el funcionamiento de esta herramienta en la figura 21.

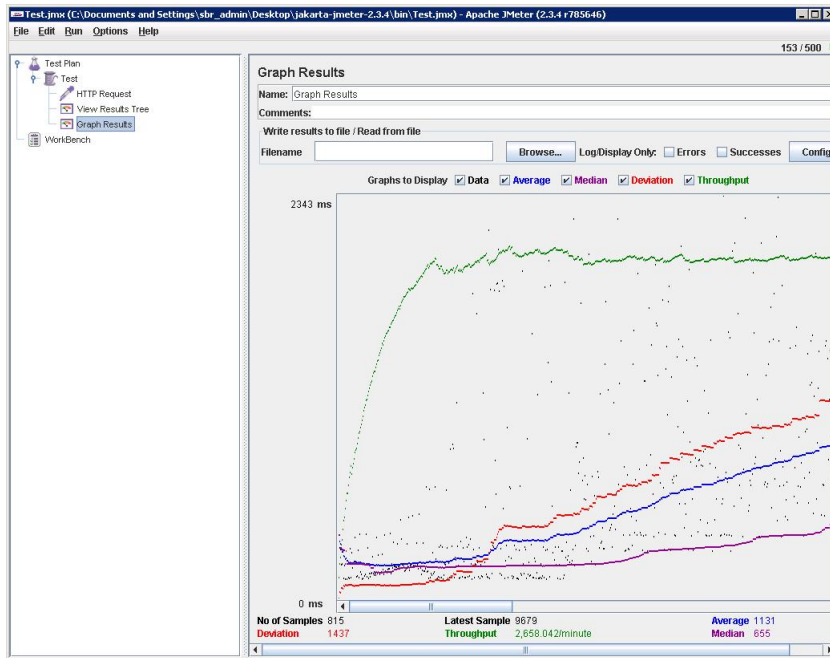


Figura 21: Resultados de Tiempo de Respuesta

11.2 PARÁMETROS DE LAS PRUEBAS

11.2.1 Servidor Web y Servidor Cache

El servidor web que utilizará será Apache, la configuración de este deberá ser:

- GNU/Linux Debian Wheezy.
- Apache2.2 con la instalación por defecto.
- 1GB de RAM
- Un núcleo
- Al menos 100MB de espacio disponible.
- Una Interfaces de Red.

11.2.2 Archivos para Realizar la Prueba

En este caso se utilizaran archivos de diferente tipo y de diferente tamaño para realizar las pruebas, entre los tipos de archivos que se utilizaran son:

- Imágenes JPEG, PNG y GIF
- Videos FLV, mpeg y AVI
- Documentos ODT, DOC y XSLT
- Instaladores EXE, BIN, MSI, DEB, RPM

Los tamaños de los archivos variarán, el máximo tamaño deberá ser de 600Mb y el mínimo de 1Mb.

11.2.3 *Estrategia de las pruebas*

Como parte de la estrategia se limitará el ancho de banda a 200 KB/s en cada servidor. Además las pruebas que se deben ejecutar son:

- Con archivos pequeños, tamaños menores a los 100Mb.
- Archivos Grandes, tamaños superiores a 100Mb.
- Por tipo de archivo.
- Todos los archivos.

11.3 ESCENARIOS

SIN WEB CACHÉ Este es el modelo tradicional de HTTP en el cual se tiene un solo servidor web y múltiples clientes se conecta a él.

PROTOTIPO DE CWC Se medirá el comportamiento de la transferencia a través del prototipo de CWC.

ANÁLISIS Y RESULTADOS

12.1 PROTOTIPO CWC

El prototipo de CWC se programó utilizando el lenguaje de programación Python. Además se utilizó una biblioteca llamada btpeer. Esta biblioteca implementa un protocolo P2P, el cual permite el intercambio de archivos. En la figura 22 se puede ver la interfaz prototipo para un cliente-servidor.

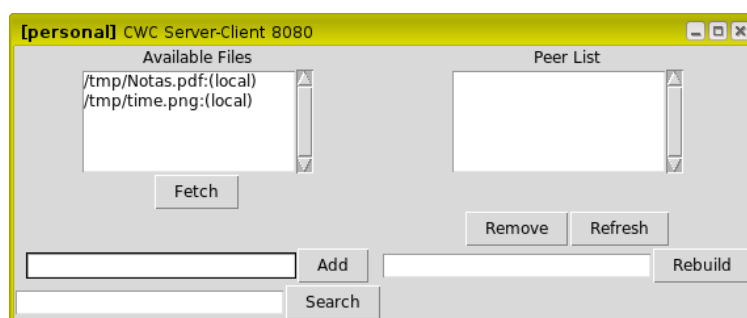


Figura 22: Prototipo de CWC

12.2 RESULTADOS DE RENDIMIENTO

TIPO DE ARCHIVO	TAMAÑO (MB)	SERVIDOR WEB
Imágenes	2.4	12.6 s
Videos	345.9	1802.1 s
Documentos	1.3	6.6s
Instaladores	34.6	181.2s

Tabla 4: Resultados del escenario Sin Web Caché

12.3 ANÁLISIS DE RENDIMIENTO DE VELOCIDAD DEL ENTORNO DE CWC

Las herramientas seleccionadas para la medición del rendimiento fueron:

TIPO DE ARCHIVO	TAMAÑO (MB)	1 NODO	2 NODOS	3 NODOS
Imágenes	2.4	13.4 s	12.9 s	13.1 s
Videos	345.9	1786.5 s	902.8 s	603.7 s
Documentos	1.3	7.1 s	6.9 s	7.0 s
Instaladores	34.6	168.4 s	88.57 s	58.2 s

Tabla 5: Resultados del escenario Prototipo CWC

- Httpperf: Se utilizó para medir el rendimiento del escenario Sin Web Caché
- time: Se utilizó para medir el rendimiento del escenario del prototipo de CWC

Basado en la tabla 4 en combinación con la tabla 5 es posible concluir que conforme más grande sea el archivo la red P2P se comporta de una mejor manera. Por el contrario, esto no sucede con los archivos pequeños, pues estos archivos son servidos siempre desde un único nodo, según el protocolo. Por otro lado se sirvieron archivos pequeños desde distintas ubicaciones, y ésto no significó mayor mejora, en cuanto al tiempo de respuesta.

12.4 RESULTADOS DE COSTO

COMPONENTE	SERVIDOR WEB	1 NODO	2 NODOS	3 NODOS
Memoria	\$40	\$40	\$0	\$0
Procesador	\$120	\$120	\$0	\$0
Almacenamiento	\$55	\$55	\$0	\$0
Interfaz de Red	\$10	\$40	\$0	\$0
Total	\$255	\$255	\$0	\$0

Tabla 6: Comparación de costos

12.5 ANÁLISIS DE COSTOS DEL ENTORNO DE CW

En el caso del CWC, basado en la tabla 6, es posible determinar que entre más nodos más barato es el mantenimiento del sitio. Esto pues, los nodos adicionales se componen de recursos que son cedidos a través de la comunidad. Por otro lado, también se puede concluir basado en la tabla 5 y la tabla 6 que el costo decrece, y el servicio aumenta, si aumentamos el número de nodos conectados a la comunidad.

Parte V

CONCLUSIONES

CONCLUSIONES Y TRABAJO FUTURO

13.1 CONCLUSIONES

- Al momento de comparar los trabajos relevantes relacionados con sistemas distribuidos se encontró que muchos otros proyectos han aunado el área de web caching distribuido, pero cada uno expone su propia necesidad, y la mayoría de ellos tiene ambiciones relacionadas con las LAN y no hay ninguno que trate de utilizar el software existente y aplicarlo en Internet directamente. Por lo que hace que el presente proyecto proponga un aporte del cual pueda beneficiar a un mayor número de personas.
- Se trató de aprovechar cualquier oportunidad de mejoramiento estratégico de las plataformas existentes, por cual sólo se agregaron algunos comandos al estándar HTTP, sin dejar la compatibilidad hacia atrás con toda la infraestructura ya creada. Promoviendo más bien el uso de módulos adicionales al servidor web y al cliente web.
- Durante el transcurso de la especificación de los requerimientos de una caché web distribuida basada en HTTP fue necesario la investigación de múltiples protocolos, modelos y análisis de distintas necesidades de mercado, para poder ajustar el nuevo modelo a un Internet cada vez más creciente. Este análisis del marco teórico arrojó mucha información que sirvió de base para la creación de esta nueva propuesta.
- En el Diseño de la arquitectura de una caché distribuida fueron notables las decisiones basadas en la transparencia de ubicación de los recursos, soluciones de bajo costo y la calidad en el servicio.

TRANSPARENCIA DE LOS RECURSOS el proyecto pretende proveer un sistema libre de fallas, totalmente distribuido el cual ponga a disposición los recursos de la comunidad formada a los miembros de la misma. Llevando el concepto de cliente servidor a otro nivel de una manera transparente.

SOLUCIÓN DE BAJO COSTO el proyecto impulsa el uso de software libre liberado bajo la licencia GPL de GNU o licencias compatibles como la licencia de Apache. Este mismo proyecto será liberado bajo estas licencias. Además impulsa

el uso de recursos subutilizados como el nivel de procesamiento de los clientes, conexión a internet, memoria, entre otros; dotándolos de un nivel más participativo en la publicación de contenidos.

CALIDAD DE SERVICIO uno de los puntos focales de este proyecto ha sido la calidad de servicio, un tema en cual están inmerso en el diseño del protocolo en sí para asegurar un servicio de alta calidad, eficaz y eficiente.

CONTINUIDAD DEL NEGOCIO se enfocaron esfuerzos para obtener un resultado que provea un mecanismo de Continuidad en el Negocio. Y es así como el protocolo CWC permite soportar caídas de los nodos y aún así seguir sirviendo el sitio web.

- En el desarrollo de un prototipo de caché distribuida, se tomaron varias decisiones de implementación una de ella fue hacer uso de tecnologías ya existentes como bibliotecas liberadas bajo licencias GPL para acelerar el proceso de desarrollo.

13.2 TRABAJO FUTURO

Se deja como trabajo futuro de el desarrollo e implementación de módulos compatibles con Apache que representen fielmente el protocolo expuesto en este documento. Además de plugins para un explorador web que cumplan la especificación del cliente cache web distribuido.

Content

13.3 EXCLUSIONES

Quedan fuera del alcance de este proyecto:

- La aplicación del protocolo de CWC en un ambiente de páginas dinámicas.
- La aplicación del protocolo de CWC en un ambiente de flujo de video o sonido.
- La seguridad en general, tanto del servidor como del cliente. No se especificó ningún mecanismo que permita la comprobación y la autenticidad de los elementos.

BIBLIOGRAFÍA

- [1] M. Ahamad, J. E. Burns, P.W. Hutto, and G Nieger. Causal memory. TR GIT-CC-93/95, Georgia Institute of Technology, July 1991.
- [2] Julio Cabero Almerana. Comunidades virtuales para el aprendizaje. su utilización en la educación. Technical report, Universidad de Sevilla.
- [3] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945 (Informational), May 1996. URL <http://www.ietf.org/rfc/rfc1945.txt>.
- [4] Brain Bershad and Matthew Zekauskas. Midway: Shared memory parallel programming with entry consistency for distributed memory multiprocessors. TR CMU-CS-91-170, Carnegie Mellon University, 1991.
- [5] Fernando Raúl Alfredo Bordinon. Diseño de una plataforma de computación distribuida cooperativa, utilizando servicios de una red compañero a compañero. *Universidad Nacional de La Plata*, 2005.
- [6] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. URL <http://www.ietf.org/rfc/rfc2616.txt>. Updated by RFCs 2817, 5785, 6266, 6585.
- [7] Michael J. Freedman, Eric Freudenthal, and David Mazières. Democratizing content publication with coral. *New York University*, 2004.
- [8] Kourosh Gharachorloo, Daniel Lenoski, James Laudon, Phillip Gibbons, Anoop Gupta, and John Hennessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. *ISCA '90 Proceedings of the 17th annual international symposium on Computer Architecture*, 18:15–26, 1990.
- [9] Sumeet Gupta and Hee Woon Kim. Virtual community: Concepts, implications, and future research directions. TR SIGEBZ02-1115, National University of Singapore, 2005.
- [10] Maurice P. Herlihy and Jeannette M. Wing. Linearizability: a correctness condition for concurrent objects. Technical report, Carnegie Mellon University.

- [11] John Hagel III and Arthur G. Armstrong. *Net Gain: Expanding Markets Through Virtual Communities*. Harvard Business School Press, 1997.
- [12] ITU. Key ict indicators for developed and developing countries and the world, 2014. URL http://www.itu.int/en/ITU-D/Statistics/Documents/statistics/2014/ITU_Key_2005-2014_ICT_data.xls.
- [13] Donald E. Knuth. Computer Programming as an Art. *Communications of the ACM*, 17(12):667–673, December 1974.
- [14] Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, pages 690–691, September 1979.
- [15] Zhengang Liang, Hossam Hassanein, and Patrick Martin. Transparent distributed web caching. *26th Annual IEEE International Conference on Local Computer Networks*, page 225, 2001.
- [16] Richard J. Lipton and Jonathan S. Sandberg. Pram: A scalable shared memory. RR 88-180, Princeton University, Department of Computer Science, 1988.
- [17] David Mosberger. Memory consistency models. TR 93/11, The University of Arizona, 1993.
- [18] Michel Raynal and Andre Schiper. From causal consistency to sequential consistency in shared memory systems. RR 2557, Institut National de Recherche en Informatique et Automatique, 1995.
- [19] J. Risson and T. Moors. Survey of research towards robust peer-to-peer networks, September 2007. URL <https://tools.ietf.org/html/rfc4981>.
- [20] Antony Rowstron, Peter Druschel, and Sitaram Iyer. Squirrel: A decentralized peer-to-peer web cache. *21th ACM Symposium on Principles of Distributed Computing*, 2001.
- [21] Jesus Salinas. Comunidades virtuales y aprendizaje digital. EDUTEC'03, IV Congreso Internacional de Tecnología Educativa y NNNT aplicadas a la Educación, Noviembre 2003.
- [22] Dheeraj Sanghi. Peer-to-peer networks and issues involving them. TR 625, Indian Institute of Technology Kanpur.
- [23] Abraham Silberschatz. *Sistemas Operativos*. Limusa Wiley, 2008.
- [24] Andrew S. Tanenbaum and D Wetherall. *Computer networks*. Pearson Prentice Hall, Boston, 5th ed edition, 2011. ISBN 9780132126953 (alk. paper).

- [25] Wathsala Vithanage, Nuwan Gunaratne, and Nishshanka Sirisena. Dalesa cache: A peer – to – peer web cache. *Lanka Software Foundation*, 2009.
- [26] Steve Whittaker, Ellen Isaacs, and Vicki O'Day. Widening the net: Workshop report on the theory and practice of physical and network communities. Technical report, SIGCHI, 1997.
- [27] Wikipedia. Caché web. http://es.wikipedia.org/wiki/Cach%C3%A9_web, 2013. URL http://es.wikipedia.org/wiki/Cach%C3%A9_web.
- [28] Wikipedia. Content delivery network. http://en.wikipedia.org/wiki/Content_delivery_network, 2013. URL http://en.wikipedia.org/wiki/Content_delivery_network.
- [29] Wikipedia. Network load balancing. http://en.wikipedia.org/wiki/Network_Load_Balancing, 2013. URL http://en.wikipedia.org/wiki/Network_Load_Balancing.
- [30] Wikipedia. P2p. <http://es.wikipedia.org/wiki/P2P>, 2013. URL http://es.wikipedia.org/wiki/P2P#Redes_P2P_centralizadas.

COLOPHON

Este documento utiliza el conjunto de tipografías de classicthesis desarrollado por André Miede. El estilo fue inspirado en la tipografía del libro de Robert Bringhurst "*The Elements of Typographic Style*"

Esta tesis fue impresa en San José, Costa Rica. Una copia de ella se puede encontrar en la siguiente dirección:

<http://www.kmoragas.com/>

Final Version as of 9 de noviembre de 2014 (classicthesis Versión 1.0).