UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA SUPERIOR EN INFORMÁTICA

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DEL SERVICIO DE ALMACENAMIENTO S3 DE AMAZON

AUTOR: MIGUEL ÁNGEL FONTICIELLA TORRE
TUTOR: FELIX GARCÍA CARBALLEIRA

Diciembre de 2009

Agradecimientos

A los que construyeron el camino que me hizo llegar hasta donde estoy y a los que me ayudarán a construir el camino que me queda por recorrer, pero sobre todo

a mi padre,

a mi madre,

y a mi hermano.

Se merecen ir en una línea cada uno. Se merecen eso y mucho más, nadie luchó tanto por mi como ellos.

Gracias.

Índice general

Capítulo 1

Introducción

En los ultimos tiempos, en el mundo de la informática las aplicaciones basadas en servicios web se están haciendo hueco a pasos agigantados. La interoperabilidad que permite dicha tecnología la convierte en un instrumento muy poderoso para la comunicación entre aplicaciones que se ejecuten en distintos equipos con distintas plataformas.

De manera muy simple podríamos describir la idea de este Proyecto Fin de Carrera (PFC) como un sistema de almacenamiento y recuperación de ficheros con una estructura cliente-servidor y basado en servicios web. El sistema está desarrollado en el lenguaje C y usa las bibliotecas de gSOAP, que más adelante explicaremos para que se usan.

Con esto, un cliente podrá guardar sus ficheros en el servidor, ejecutado en otra máquina, para despues poder recuperarlos cuando quiera, de manera totalmente transparente y sin importar las plataformas. El servidor ofrecerá al cliente una serie de operaciones, definidas en una interfaz y respetando en todo momento el estandar SOAP.

El producto comercial en el que se basa este PFC es Amazon S3. Aunque no pretende cubrir el 100 % de las funcionalidades de este, sí que se desea cubrir aquellas funcionalidades básicas relacionadas con el almacenamiento y la recuperación de objetos. A lo largo de este documento profundizaremos en las similitudes y diferencias que presenta este PFC con la aplicación Amazon S3.

1.1. Tecnologías relacionadas con el proyecto

En esta sección repasaremos una serie de tecnologías que conviene conocer antes de tratar a fondo el proyecto, como son Amazon S3, los servicios web, SOAP, gSOAP y WSDL.

1.1.1. Amazon S3

Amazon S3 es un servicio web de almacenamiento on line orientado a desarrolladores ofrecido por Amazon. Amazon S3 provee almacenamieno ilimitado a través de una interfaz simple. Este servicio no es gratuito. Amazon establece un cargo de 0.15 dolares por gigabyte al mes, mas otros cargos adicionales dependiendo del ancho de banda usado para recuperar los objetos almacenados.

Muchas empresas contratan este servicio para alojar el contenido de su web, olvidándose de todo y obteniendo un servicio eficaz y relativamente barato. Ejemplos destacados son Twitter, el popular servicio de microblogging, que lo utiliza para la alojar las fotografías del perfil de sus usuarios. Otras, como Vimeo, una de las principales plataformas destinadas a la emisión en streaming de vídeo, se sirve de los servicios de S3 para almacenar los vídeos subidos por los usuarios.

Los detalles del diseño de Amazon S3 no son públicos, pero lo que sí se ofrece es su interfaz para que el desarrollador la pueda usar en sus aplicaciones. Esta interfaz es la base del proyecto que nos ocupa. En torno a ella se construyen un cliente y un servidor cuyo punto de unión es dicha interfaz.

Las funcionalidades mas destacadas de Amazon S3 son:

- Escribir, leer y borrar objetos de un tamaño maximo de 5GB. El número de objetos que el usuario puede guardar es ilimitado.
- Cada objeto se guarda en un bucket asignándosele una clave única.
- Provee mecanismos de autenticación para asegurar una transferencia y almacenamiento seguro.

- Los objetos pueden ser públicos o privados y el usuario podrá definir los privilegios de sus objetos para que otros usuarios puedan acceder a ellos.

- Ofrece dos interfaces con las que el usuario podrá interactuar: REST y SOAP.

Puesto que el objetivo del proyecto es desarrollar un servicio similar a Amazon S3, no vamos a entrar más en detalle en este apartado y cuando se describa una funcionalidad concreta del proyecto se citarán las principales similitudes y diferencias con Amazon S3.

1.1.2. Introducción a los servicios Web

Un servicio web (en inglés Web service) es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos. Las organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los servicios Web. Para mejorar la interoperabilidad entre distintas implementaciones de servicios Web se ha creado el organismo WS-I, encargado de desarrollar diversos perfiles para definir de manera más exhaustiva estos estándares. Más adelante profundizaremos en el marco en el que se desarrollan actualmente los servicios web

Estándares

A continuación se citan los estándares más importantes de los que hacen uso los servicios web:

- Web Services Protocol Stack: Así se denomina al conjunto de servicios y protocolos de los servicios Web.

- XML (Extensible Markup Language): Es el formato estándar para los datos que se vayan a intercambiar.

- SOAP (Simple Object Access Protocol) o XML-RPC (XML Remote Procedure Call): Protocolos sobre los que se establece el intercambio.

- Otros protocolos: los datos en XML también pueden enviarse de una aplicación a otra mediante protocolos normales como HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), o SMTP (Simple Mail Transfer Protocol).

- WSDL (Web Services Description Languages): Es el lenguaje de la interfaz pública para los servicios Web. Es una descripción basada en XML de los requisitos funcionales necesarios para establecer una comunicación con los servicios Web.

- DDI (Universal Description, Discovery and Integration): Protocolo para publicar la información de los servicios Web. Permite comprobar qué servicios web están disponibles.

- WS-Security (Web Service Security): Protocolo de seguridad aceptado como estándar por OASIS (Organization for the Advancement of Structured Information Standards). Garantiza la autenticación de los actores y la confidencialidad de los mensajes enviados...

Ventajas de los servicios Web

- Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.

- Los servicios Web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.

- Al apoyarse en HTTP, los servicios Web pueden aprovecharse de los sistemas de seguridad firewall sin necesidad de cambiar las reglas de filtrado.

- Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.

- Permiten la interoperabilidad entre plataformas de distintos fabricantes por medio de protocolos estándar y abiertos. Las especificaciones son gestionadas por una organización abierta, la W3C, por tanto no hay secretismos por intereses particulares de fabricantes concretos y se garantiza la plena interoperabilidad entre aplicaciones.

Inconvenientes de los servicios Web

- Para realizar transacciones no pueden compararse en su grado de desarrollo con los estándares abiertos de computación distribuida como CORBA (Common Object Request Broker Architecture).

- Su rendimiento es bajo si se compara con otros modelos de computación distribuida, tales como RMI (Remote Method Invocation), CORBA, o DCOM (Distributed Component Object Model). Es uno de los inconvenientes derivados de adoptar un formato basado en texto. Y es que entre los objetivos de XML no se encuentra la concisión ni la eficacia de procesamiento.

- Al apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en firewall cuyas reglas tratan de bloquear o auditar la comunicación entre programas a ambos lados de la barrera. Por otra parte también puede ser una ventaja, como veremos a continuación.

Razones para crear servicios Web

La principal razón para usar servicios Web es que se basan en HTTP sobre TCP (Transmission Control Protocol) en el puerto 80. Dado que las organizaciones protegen sus redes mediante firewalls -que filtran y bloquean gran parte del tráfico de Internet-, cierran casi todos los puertos TCP salvo el 80, que es, precisamente, el que usan los navegadores. Los servicios Web utilizan este puerto, por la simple razón de que no resultan bloqueados.

Otra razón es que, antes de que existiera SOAP, no había buenas interfaces para acceder a las funcionalidades de otros ordenadores en red. Las que había eran ad hoc y poco conocidas, tales como EDI (Electronic Data Interchange), RPC (Remote Procedure Call), u otras APIs.

Una tercera razón por la que los servicios Web son muy prácticos es que pueden aportar gran independencia entre la aplicación que usa el servicio Web y el propio servicio. De esta forma, los cambios a lo largo del tiempo en uno no deben afectar al otro. Esta flexibilidad será cada vez más importante, dado que la tendencia a construir grandes aplicaciones a partir de componentes distribuidos más pequeños es cada día más utilizada.

Se espera que para los próximos años mejoren la calidad y cantidad de servicios ofrecidos basados en los nuevos estándares.

1.1.3. SOAP

Son las siglas de Simple Object Access Protocol. Este protocolo deriva de un protocolo creado por David Winer, XML-RPC en 1998. En su sitio web, Userland (http://www.userland.com) se puede encontrar multitud de documentación acerca de este primer protocolo de comunicación bajo http mediante XML. Con este protocolo se pedían realizar RPC o remote procedure calls, es decir, podíamos bien en cliente o servidor realizar peticiones mediante http a un servidor web. Los mensajes debían tener un formato determinado empleando XML para encapsular los parámetros de la petición. Con el paso del tiempo el proyecto iniciado por David Winer interesó a importantes multinacionales entre las que se encuentran IBM y Microsoft y de este interés por XML-RPC se desarrollo SOAP.

En el núcleo de los servicios web se encuentra el protocolo simple de acceso a datos SOAP, que proporciona un mecanismo estándar de empaquetar mensajes. SOAP ha recibido gran atención debido a que facilita una comunicación del estilo RPC entre un cliente y un servidor remoto. Pero existen multitud de protocolos creados para facilitar la comunicación entre aplicaciones, incluyendo RPC de Sum, DCE de Microsoft, RMI de Java y ORPC de CORBA. ¿Por qué se presta tanta atención a SOAP?

Una de las razones principales es que SOAP ha recibido un increíble apoyo por parte de la industria. SOAP es el primer protocolo de su tipo que ha sido aceptado prácticamente por todas las grandes compañías de software del

mundo. Compañías que en raras ocasiones cooperan entre sí están ofreciendo su apoyo a este protocolo. Algunas de las mayores compañías que soportan SOAP son Microsoft, IBM, SUN, Microsystems, SAP y Ariba.

Algunas de las ventajas de SOAP son:

- No está asociado con ningún lenguaje. Los desarrolladores involucrados en nuevos proyectos pueden elegir desarrollar con el último y mejor lenguaje de programación que exista pero los desarrolladores responsables de mantener antiguas aflicciones heredadas podrían no poder hacer esta elección sobre el lenguaje de programación que utilizan. SOAP no especifica una API, por lo que la implementación de la API se deja al lenguaje de programación, como en Java, y la plataforma como Microsoft .Net.

- No se encuentra fuertemente asociado a ningún protocolo de transporte. La especificación de SOAP no describe como se deberían asociar los mensajes de SOAP con HTTP. Un mensaje de SOAP no es más que un documento XML, por lo que puede transportarse utilizando cualquier protocolo capaz de transmitir texto.

- No está atado a ninguna infraestructura de objeto distribuido. La mayoría de los sistemas de objetos distribuidos se pueden extender, y ya lo están alguno de ellos para que admitan SOAP.

- Aprovecha los estándares existentes en la industria. Los principales contribuyentes a la especificación SOAP evitaron, intencionadamente, reinventar las cosas. Optaron por extender los estándares existentes para que coincidieran con sus necesidades. Por ejemplo, SOAP aprovecha XML para la codificación de los mensajes, en lugar de utilizar su propio sistema de tipo que ya están definidas en la especificación esquema de XML. Y como ya se ha mencionado SOAP no define un medio de trasporte de los mensajes; los mensajes de SOAP se pueden asociar a los protocolos de transporte existentes como HTTP y SMTP.

- Permite la interoperabilidad entre múltiples entornos. SOAP se desarrolló sobre los estándares existentes de la industria, por lo que las aplicaciones que se ejecuten en plataformas con dicho estándares pueden

comunicarse mediante mensaje SOAP con aplicaciones que se ejecuten en otras plataformas. Por ejemplo, una aplicación de escritorio que se ejecute en una PC puede comunicarse con una aplicación del back-end ejecutándose en un mainframe capaz de enviar y recibir XML sobre HTTP.

gSOAP

gSOAP es una herramienta opensource que facilita la implementación del protocolo SOAP sobre XML usando como lenguaje C/C++. Provee al programador, de una forma transparente, una API para crear mensajes SOAP fácilmente.

1.1.4. WSDL

WSDL son las siglas de Web Services Description Language, un formato XML que se utiliza para describir servicios Web (algunas personas lo leen como 'wisdel'). La versión 1.0 fue la primera recomendación por parte del W3C y la versión 1.1 no alcanzó nunca tal estatus. La versión 2.0 se convirtió en la recomendación actual por parte de dicha entidad.

WSDL describe la interfaz pública de los servicios Web. Está basado en XML y describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje.

Así, WSDL se usa a menudo en combinación con SOAP y XML Schema. Un programa cliente que se conecta a un servicio web puede leer el WSDL para determinar que funciones están disponibles en el servidor. Los tipos de datos especiales se incluyen en el archivo WSDL en forma de XML Schema. El cliente puede usar SOAP para hacer la llamada a una de las funciones listadas en el WSDL.

La relacion entre este PFC y WSDL se reduce a un punto: la interfaz pública que ofrece Amazon S3 y que es la base de partida para este proyecto está definida en WSDL. Si se respeta esta interfaz se conseguirá el objetivo

del proyecto.

1.2. Objetivos del proyecto

El objetivo de este PFC en un principio es desarrollar un servidor con unas características similares a las de Amazon S3. Evidentemente, no pretende ser un clon de Amazon, porque los recursos no son comparables, por lo que solo se implementará una parte con los servicios básicos.

En un principio se empezó desarrollando el servidor, probandolo con clientes de Amazon S3 disponibles en la web oficial, pero posteriormente y para facilitar el desarrollo de partes más complejas de la aplicación, se optó por el desarrollo de forma paralela de un cliente que usará la funcionalidad ofrecida por el servidor.

Ambos, tanto cliente como servidor, han sido desarrollados en el lenguaje de programacion C y usando las herramientas que proporciona gSOAP, que nos permitirá componer mensajes SOAP con los que se comunicarán las aplicaciones cliente y servidor.

Aunque el lenguaje del cliente y del servidor en este caso es el mismo, una de las ventajas de usar el estándar SOAP es que es independiente del lenguaje de programación y, por lo tanto, clientes desarrollados en un lenguaje distinto al del servidor podrán interactuar con el sin ningún problema.

Al hilo de esto, comentar que Amazon S3 ofrece dos interfaces al usuario: una interfaz SOAP, que será la que se implemente en este PFC, y otra REST, con las mismas funcionalidades pero que queda fuera de este proyecto.

Para dar una idea general de las capacidades que abarca este PFC, adelantamos que las funcionalidades que implementará el servidor son las siguientes:

- Crear/borrar/compartir buckets
- Subir/borrar/recuperar/copiar/compartir objeto
- Listar todos los buckets de un usuario
- Listar los objetos que contiene un determinado bucket

- Mecanismos de autenticacion basados HMAC-SHA1

Además de esto, el cliente añadirá nuevas funcionalidades basadas en las anteriores, esto es, combinando dos o mas llamadas a la interfaz ofrecida por el servidor. Estas funcionalidades añadidas son:

- Buscar objeto
- Mover objeto
- Subir/bajar bucket
- Listar todo

Con estas funcionalidades básicas se pretende crear un sistema que sin ser todo lo complejo que es Amazon S3, tengo una cierta utilidad.

1.3. Estructuración del documento

Para poner al lector en antecedentes, pasamos a hacer una descripcion de lo que se va a encontrar en esta memoria:

- En primer lugar haremos una breve reseña al estado del arte en el que nos adentramos, concretamente el estado del arte de los sistemas distribuidos, el almacenamiento en dichos sistemas, "cloud computing" y los servicios web.
- Pasaremos luego a la sección donde explicaremos el diseño de la aplicación comentando sus rasgos mas característicos.
- Después de todo diseño tiene lugar una implementación del mismo. En este caso veremos los pormenores de las distintas operaciones y como se implementan en el lado del servidor y en el lado del cliente.
- A continuación veremos las pruebas realizadas, comparando la aplicación en términos de velocidad con otras aplicaciones de transferencia de datos.
- Tras la evaluación extraeremos una serie de conclusiones en las que se hará un repaso de los problemas encontrados y los resultados obtenidos, entre otras cosas.

- Un apartado importante es el que contempla las líneas futuras, que son posibles mejoras o ampliaciones de la aplicacion que ya sea por falta de tiempo o de recursos no se han podido llevar a cabo pero que resultan interesantes para completar el proyecto.
- Veremos también un capítulo dedicado al presupuesto del proyecto donde, como en todo presupuesto, se especificarán los gastos estimados que, en este caso, conllevaría el desarrollo del proyecto.
- Los próximos tres capítulos estan dedicados a la informacion referente al entorno en el que se llevo a cabo el desarrollo, citando la bibliografía, el software usado y las librerías externas en las que nos hemos apoyado.
- Por último y no por ello menos importante, se incluye un anexo con un completo manual del cliente para que el usuario sepa como sacarle el máximo partido.

Capítulo 2

Estado del arte

En esta sección trataremos de presentar el estado actual de los distintos campos que toca el proyecto, comentando brevemente las principales líneas que se están siguiendo actualmente en el desarrollo de aplicaciones relacionadas con dichos campos.

2.1. Sistemas distribuidos

Un sistema distribuido se define como una colección de computadores separados físicamente y conectados entre sí por una red de comunicaciones distribuida; cada máquina posee sus componentes de hardware y software que el usuario percibe como un solo sistema (no necesita saber qué cosas están en qué máquinas). El usuario accede a los recursos remotos (RPC) de la misma manera en que accede a recursos locales.

Los sistemas distribuidos deben ser muy confiables, ya que si un componente del sistema se descompone, otro componente debe de ser capaz de reemplazarlo, esto se denomina "Tolerancia a Fallos".

El tamaño de un sistema distribuido puede ser muy variado, ya sean decenas de hosts (red de área local), centenas de hosts (red de área metropolitana), y miles o millones de hosts (Internet); esto se denomina "Escalabilidad".

Los sistemas distribuidos necesitan de un soporte de comunicaciones. Este servicio de comunicaciones debe ser fiable y con un rendimiento aceptable. Además, en los sistemas distribuidos aparecen los problemas clásicos de los sistemas concurrentes: recusos compartidos, sincronización, etc.

Vemos ahora las propiedades de los sistemas distribuidos.

2.1.1. Propiedades

Los sistemas distribuidos deben intentar proporcionar transparencia, eficiencia, flexibilidad, escalabilidad y fiabilidad. Sin embargo, estos aspectos son, en parte, contrarios y, por lo tanto, al diseñar un sistema distribuido se debe intentar cumplir de manera aceptable con cada uno de ellos:

- Transparencia. Es labor del sistema establecer los mecanismos que oculten la naturaleza distribuida del sistema y que permitan trabajar a los usuarios como si de un único equipo se tratara.
- Eficiencia. La idea base de los sistemas distribuidos es obtener sistemas que, estando formados por un conjunto de ordenadores, sean mucho más rápidos que cualquiera de estos ordenadores por separado. En la práctica, esto es una utopía. El coste asociado a la comunicación de las distintas máquinas que componen el sistema distribuido hace que sus prestaciones disminuyan de forma considerable.

  Para lograr que un sistema como este sea eficiente hay que poner en práctica la idea de ejecutar un programa en un único procesador del sistema y, entonces, distribuir las tareas a realizar por éste en varios procesadores; necesitándose, por tanto, nuevas herramientas que permitan desarrollar aplicaciones de este tipo.
- Flexibilidad de los sistemas operativos. Un campo en constante desarrollo como es el diseño de sistemas operativos distribuidos, debe estar abierto a cambios y actualizaciones constantes que mejoren su funcionamiento. Esta necesidad ha generado dos posibles arquitecturas para el núcleo del sistema operativo: el núcleo monolítico y el micronúcleo. Las diferencias fundamentales entre ambos son los servicios que ofrece. Mientras el núcleo monolítico ofrece todas las funciones básicas del sistema, el micronúcleo incorpora solamente las fundamentales; como son, control y comunicación entre procesos, y gestión de la memoria. El resto de servicios se cargan dinámicamente en función de las demandas del usuario.
- Escalabilidad. Un sistema distribuido debería funcionar de igual forma tanto para unos pocos ordenadores como para un conjunto grande de

ellos. Igualmente, debería no ser determinante el tipo de red utilizada (LAN o WAN) ni las distancias físicas entre los equipos que la conforman. Aunque esto sería lo deseable, en la práctica, no ocurre. Del mismo modo, el tipo de red condiciona tremendamente el rendimiento del sistema; por tanto, puede que lo que funcione para un tipo de red, para otro requiera un nuevo diseño.

- Fiabilidad. Una de las ventajas claras que ofrece la idea de un sistema distribuido, es que el funcionamiento del sistema no debe estar ligado a ciertas máquinas, sino que cualquier equipo pueda suplir a otro en caso de que uno se estropee o falle. La forma más evidente de lograr la fiabilidad de todo el sistema es el uso de redundancia, es decir, la información no debe estar almacenada en una sola máquina, sino en un conjunto de ellas.

2.1.2. Campos de aplicación

En relación con los campos de aplicación de este tipo de sistemas podemos distinguir, por un lado, aquellos donde la distribución es fundamentalmente un medio para conseguir un fin y, por otro, aquellos donde es un problema en sí mismo.

En los primeros, el uso de soluciones distribuidas pretende alcanzar las siguientes metas:

- Computación masivamente paralela, de propósito general y de alta velocidad.
- Tolerancia a fallos (confianza, disponibilidad).
- Respuesta a demandas con requisitos de tiempo real.

En los segundos, son los propios requisitos de la aplicación los que fuerzan a evolucionar hacia soluciones distribuidas:

- Bases de datos o sistemas de archivos distribuidos. Es necesario acceder a los datos desde lugares geográficamente dispersos y, además, puede ser también conveniente (e incluso imprescindible) almacenarlos en varios lugares diferentes manteniendo la consistencia de los mismos.
- Fabricación automatizada. Es necesaria la colaboración de muchos procesadores para coordinar las tareas a desempeñar.

- Supervisión remota y control. Los puntos (sensores, actuadores, nodos) donde se toman las decisiones de control pueden estar diseminados en diferentes partes de un sistema distribuido.
- Toma de decisiones coordinada. Hay muchas aplicaciones donde es necesario que varios procesadores participen en la toma de decisiones, por ejemplo, porque cada uno de ellos tiene una parte relevante de los datos y es necesario fusionarlos en cualquier momento.

Los sistemas distribuidos necesitan obligatoriamente un soporte de comunicaciones. Este servicio de comunicaciones debe ser fiable y presentar un rendimiento aceptable. Además, en los sistemas distribuidos aparecen los problemas clásicos de los sistemas concurrentes: recursos compartidos y sincronización, entre otros. En las siguientes secciones se van a analizar estas características.

2.1.3. Comunicación y sincronización

El proceso es el concepto básico para expresar concurrencia, tanto en un sistema centralizado como en uno distribuido. Con la introducción de los procesos se permite dividir el software en módulos, cada uno de los cuales expresa una actividad lógicamente concurrente con el resto. Este modelo permite mayor simplicidad y más fácil comprensión del software utilizado, frente al comportamiento, muchas veces confuso, de un programa secuencial que trata de expresar un comportamiento concurrente.

El modelo de proceso resulta incompleto si no se proporcionan mecanismos para la comunicación entre procesos. Esto es debido a que, normalmente, en todo sistema concurrente, existen interacciones entre sus diferentes actividades concurrentes.

La diferencia más importante entre un sistema distribuido y uno centralizado es precisamente cómo se realiza la comunicación entre procesos. En un sistema centralizado, las interacciones entre procesos se hacen sobre memoria compartida, mientras que en un sistema distribuido se necesita intercambio de mensajes. Los mecanismos a estudiar son entonces: el paso de mensajes punto a punto y la llamada remota a procedimiento. Una variante de los mensajes punto a punto son los buzones. Estos se usan para enviar o recibir mensajes, permitiendo no sólo desacoplar emisor y receptor sino, también, tener múltiples emisores y receptores.

En un sistema distribuido, un emisor también debería poder referenciar, al mismo tiempo, a un conjunto de receptores ya que la información contenida en un mensaje puede ser necesaria en más de un punto del sistema distribuido. Por tanto, la comunicación con grupos de procesos es otro tipo de interacción que conviene analizar y solucionar. Los grupos de procesos son una herramienta natural para programar un sistema distribuido, y su importancia ha ido incrementándose en entornos de programación distribuida a lo largo del tiempo.

2.1.4. Tolerancia a fallos

Un sistema distribuido es una colección de nodos autónomos de computación que se pueden comunicar unos con otros y que colaboran en un objetivo o tarea común. Cuando un nodo falla o cuando el subsistema de comunicaciones que permite que los nodos se comuniquen falla, los nodos se han de adaptar a las nuevas condiciones, para que puedan seguir cooperando.

En los sistemas tolerantes a fallo, los fallos que se pueden tolerar son aquéllos que está previsto que pueden ocurrir. El primer paso necesario para que un sistema pueda recuperarse de un fallo, es detectarlo. El siguiente paso es llevar al sistema a un estado consistente, para ello, es necesario que las acciones realizadas antes del fallo mantengan la consistencia. La clave para tolerar fallos es la replicación, es decir, que varios elementos del sistema puedan dar el mismo servicio.

Los sistemas de computación constan de multitud de componentes hardware y software que pueden fallar (debido a un error de diseño, al envejecimiento de los materiales o a un evento externo). En muchos sistemas, estos fallos pueden llegar a producir inconsistencias y, por lo tanto, la no disponibilidad del servicio que estaban ofreciendo. Existen sistemas que se diseñan para tolerar fallos. Estos sistemas (sistemas tolerantes a fallos) hacen transparente al usuario los modos de fallos previstos en el sistema.

Cuando se quiere construir un sistema tolerante a fallos, habitualmente se consideran dos alternativas. La primera consiste en ejecutar sistemas software sobre hardware especializado tolerante a fallos (CPU y memoria principal replicadas, discos espejo, varios buses y rutas de datos, entre otros.) pero; debido a razones económicas, fundamentalmente, la segunda aproximación está ganando impulso y es la de usar hardware estándar para soportar tolerancia a fallos, replicando ese hardware y manteniendo esa replicación a nivel

software. De esta forma se consigue un sistema distribuido tolerante a fallos.

2.1.5. Arquitectura cliente-servidor

La arquitectura cliente-servidor es una da las más utilizadas para el desarrollo de aplicaciones distribuidas. En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema.

La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa. Los tipos específicos de servidores incluyen los servidores web, los servidores de archivo, los servidores del correo, etc. Mientras que sus propósitos varían de unos servicios a otros, la arquitectura básica seguirá siendo la misma.

2.2. Servicios Web

Ya hemos explicado un poco por encima en la introducción de este Proyecto Fin de Carrera lo que son los "Servicios Web", ahora vamos a profundizar en los principales usos y tendencias dentro del mundo de la programación que usan dichos servicios.

Para comenzar vamos a ver el diagrama que representa la pila de protocolos que intervienen en un "Servicio Web" (figura 2.1).

La capa de más abajo representa el protocolo básico de comunicación y sobre esta se van añadiendo el resto de capas, que incorporan abstracciones de más alto nivel. Las capas más altas no dependen necesariamente de la anterior, pero se representan así para simplificar el visionado del esquema.

2.2.1. Tecnología de los Servicios Web

Servicio de transporte

La principal función de este servicio es transferir datos desde una máquina hasta otra. Los principales protocolos usados en esta capa son HTTP, SMTP

Layer Description	Implementation(s)	Other Concerns			
Standard Messaging	Electronic Business XML Initiative (ebXML)	Quality of Service	Management	Security	Service Development
Service Composition	Business Process Execution Service for Web Services (BPEL4WS)				
Service Registry	Universal Description, Discovery and Integration (UDDI) ebXML Registries				
Service Description	Web Services Description Language (WSDL)				
Service Messaging	Simple Object Access Protocol (SOAP)/Extensible Markup Language (XML)				
Service Transport	Hypertext Transfer Protocol (HTTP) Simple Mail Transfer Protocol (SMTP) File Transfer Protocol (FTP)				

Figura 2.1: Pila de protocolos de un "Servicio Web"

y FTP. El más extendido de ellos es HTTP, que a su vez es el más usado en Internet. Esto no es casualidad. Los "Servicios Web" se basan en este protocolo puesto que no es bloqueado por la mayoría de los firewalls. Además, HTTP 1.0 es un protocolo no orientando a conexión cuyos mensajes son autodescriptivos, no necesitan de un establecimiento de sesión ni de ningun tipo de conversación entre cliente y servidor.

Servicio de mensajes

Esta capa describe el formato de los datos usado para trasnferirlos por la red. Este formato está basado en XML, que es un lenguaje de marcado que es capaz de representar los datos de manera estructurada y transferirlo como texto. Esto lo usan los servicios web para la especificación de los servicios prestados (nombre de servicio, parametros recibidos, resultados devueltos...) así como para realizar las llamadas remotas a dichos servicios.

SOAP (Simple Object Access Protocol) no es otra cosa que un protocolo estándar que especifica cómo construir los mensajes en XML para usar un servicio web.

Descripción de servicios

Esta capa especifica tres aspectos fundamentales del servicio web:

- Operaciones ofrecidas por el servicio web

- Mensajes que el servicio web aceptará
- El protocolo que el consumidor del servicio debe vincularse para acceder al sistema

El servicio web usa WSDL (Web Services Description Language) para especificar un contrato de servicio. Este contrato es una descripción de la entidad encargada de recibir el mensaje generado así como el formato de dicho mensaje XML. Estas entidades receptoras son direcciones de red que aceptan el formato de mensaje XML especificado en el contrato de servicio.

Normalmente, para hacer todo este proceso de comunicación más transparente al cliente se generan de manera estática unos "stubs" que ponen al servicio del programador una interfaz que por debajo se encarga de la generación de mensajes XML con el formato acordado. En este PFC es el framework gSOAP el que se encarga de esta abstracción para crear mensajes acordes al WSDL que nos hemos descargado de Amazon y que sera la interfaz de nuestro servicio web. Otras tecnologías generan estos "stubs" de manera dinámica en tiempo de ejecución.

Registro del servicio

Los servicios web soportan el concepto de "Descubrimiento Dinámico de Servicios". Un consumidor de servicios usa un registro para averiguar cuáles son los servicios disponibles. Existe un Servicio Web llamado UDDI (Universal Description, Discovery and Integration) que proporciona al consumidor información sobre los servicios web disponibles así como su ubicacion. UDDI es uno de los estándares básicos de los servicios web cuyo objetivo es ser accedido por los mensajes SOAP y dar paso a documentos WSDL, en los que se describen los requisitos del protocolo y los formatos del mensaje solicitado para interactuar con los servicios web.

Composición del servicio

La compososición del servicio es un termino emergente actualmente y que se refiere a la manera de combinar distintos servicios web dentro de un proceso de negocio. Sería algo así como una "orquestación" de servicios. Esto se vería como una conversación entre los servicios Web que formaría una gran transacción, cuyo éxito depende del éxito de todos y cada uno de las llamadas

a servicios.

Hay varios lenguajes que especifican estos flujos de negocio. Los dos mas populares son WSFL y XLANG. Recientemente IBM, Microsoft y BEA combinaron estas dos especificaciones para elaborar otra llamada BPEL4WS (Business Process Execution Language for Web Services).

2.2.2. Escenarios de aplicacion de los Servicios Web

Los Servicios Web se pueden usar en un amplio rango de aplicaciones, desde aplicaciones domésticas hasta videojuegos, ya que estos servicios resuelven un gran número de problemas. No obstante, las dos aplicaciones prácticas en las que se estan abriendo camino más rapidamente son en la construcción de aplicaciones B2B (business-to-business) y EAI (Enterprise Application Integration).

B2B

Las aplicaciones B2B son aquellas que se encargan de la comunicación entre empresas. Los servicios web son muy interesantes para construir este tipo de aplicaciones debido a la heterogeneidad que presentan las plataformas y estándares de cada empresa. Dos estándares que usan servicios web para especificar los detalles de la comunicación en el comercio electrónico son ebXML (www.ebXML.org) y RosettaNet. Antes de estos existía el estandar EDI (Electronic Data Interchange), pero resulta demasiado caro y dependiente de formatos de datos propietarios y diseñados para un determinado tipo de actividad. Estandares como ebXML permiten a los desarrolladores construir aplicaciones basadas en estructuras y sintaxis comunes para comunicarse entre todo tipo de empresas. Evidentemente a partir de esto, se crean consorcios de empresas que desarrollan especificaciones propias de un determinado sector como pueden ser la banca o las compañías de seguros.

Los “Servicios Web” no proveen por ellos mismos capacidades de colaboración entre los procesos de la empresa, sino que simplemente definen unas bases para que las organizaciones puedan comunicarse entre ellas de manera más compleja, permitiendo a otros socios comerciales participar en un proceso de negocios abierto y complejo. Proyectos como ebXML, además, han estandarizado servicios comunes como pueden ser excepciones, seguridad y notificación de fallos.

EAI

Las aplicaciones EAI son una solución al problema de comunicar diversas aplicaciones que en un principio no tienen ningun tipo de organizacion. Las EAI convierten los datos y formatos utilizados internamente de manera que todas se "entiendan". Esto es básicamente el problema que resuelven los servicios web.

La topología típica de los sistemas EAI es la conocida como hub-and-spoke. En ella, cuando una aplicacion necesita comunicarse con otra le manda el mensaje al EAI y este lo transforma y lo enruta según proceda. En cambio, cuando se quiere implementar un EAI usando "Servicios Web" se prescinde de la topología hub-and-spoke y se usa otra en la que el servidor del servicio se comunica directamente con el cliente.

2.3. Almacenamiento en sistemas distribuidos

Este tipo de sistemas de almacenamiento nació en la década de 1990, aunque hoy en día está adquiriendo una gran relevancia. Esto se debe principalmente a su expansión del ámbito doméstico al ámbito empresarial, donde su uso era muy restringido hasta hace bien poco. Las grandes empresas suelen utilizar aplicaciones propias para centralizar la información en servidores, de tal manera que desde cualquier unidad dentro de su red sea más sencillo acceder a la información. No obstante, esta tendencia se esta invirtiendo debido a las necesidades de computo y de almacenamiento de las empresas, a las cuáles les sale mas barato contratar estos servicios que hacerse con las infraestructuras necesarias para satisfacer sus necesidades. Aqui es donde entran en juego los servicios web, y en el caso que nos importa Amazon S3.

Como los sistemas de almacenamiento distribuido que nos interesan son de tipo "Cloud Computing", en el siguiente apartado profundizaremos un poco más en las ventajas que aportan.

2.4. Cloud Computing

El "Cloud Computing" (computación en nube) es un tema muy de moda en estos tiempos. Para algunos representa un avance y para otros una vuelta

a los orígenes. Para entender mejor esta polémica vamos a explicar brevemente que significa este concepto.

El actual "cloud computing" consiste en crear sistemas informáticos centralizados donde las empresas de nueva creación puedan disponer de un espacio de almacenamiento, ancho de banda y tiempo de computación con un ahorro importante de costes, al no tener la necesidad de disponer de una gran infraestructura para escalar la plataforma (es decir, para hacerla más grande a medida que crezca la demanda del servicio que ofrezcan). Así, se puede hablar de informática de alquiler.

2.4.1. Ventajas que ofrece

Las ventajas para las empresas radican principalmente en poder escalar rápidamente, sin tener que adquirir hardware y software adicional. Si una aplicación crece de forma exponencial en recursos, la plataforma lo hará automáticamente. Así, una PYME o un programador independiente puede crear una aplicación que pueda ser usada en todo el mundo sin necesidad de invertir en infraestructuras para sustentar la demanda. De esta manera se minimiza la posibilidad de "morir de éxito"

Los usuarios de estos servicios pagan por la cantidad de recursos asignados, el tiempo de computación y la capacidad de almacenamiento utilizada, ahorrando tanto en la compra de tecnología como en consumo energético. Esto les permite centrarse en crear aplicaciones sin preocuparse de más problemas.

Empresas privadas de cualquier tamaño, administraciones públicas, universidades y otras instituciones son algunos de los sectores donde la utilización del "cloud computing" puede suponer una ventaja al sustituir a las redes internas propias por una infraestructura distribuida basada en protocolos abiertos.

Sin embargo, los primeros clientes de estos servicios son las empresas de la Web 2.0, también llamada Internet Social, quienes ven una oportunidad de lanzar una aplicación de forma rápida sin tener que hacer grandes inversiones.

2.4.2. Similitudes y diferencias con la computación distribuida

A pesar de las similitudes, el "cloud computing" no es el "grid computing", o computación distribuida. Es decir, la utilización de tiempo de proceso en un supercomputador o en una red de ordenadores distribuidos. El "cloud computing" tiene más relación con la creación de aplicaciones que funcionan gracias a diferentes tipos de procesos (que pueden ser ejecutados desde cualquier dispositivo y lugar) y al almacenamiento de datos en diferentes servidores distribuidos.

Dentro de este mismo concepto de "computación en la nube" también cabe situar la creación de software para ser ofrecido como servicio. Numerosas empresas de la llamada Web 2.0 trabajan de esta forma, con lo que permiten que los usuarios accedan a aplicaciones comunes a través de Internet, pero que no tienen por qué funcionar desde los ordenadores de la compañía. Así, como ya hemos comentado, el sistema de imágenes de los usuarios en Twitter no está alojado en sus servidores, sino que corre por cuenta del servicio Amazon Web Services.

2.4.3. Sistemas operativos centrados en la red

Hasta ahora, con el dominio de las aplicaciones pensadas para ser ejecutadas desde el escritorio de los usuarios, los sistemas operativos representaban el eje central de la actividad informática. Los programas y datos principales estaban almacenados en el ordenador del usuario y las actualizaciones y parte de los últimos datos se encontraban en Internet. Por eso se decía que los sistemas operativos, sobre todo los privativos como Windows y Mac OS X, eran centrados en el PC.

Sin embargo, con el "cloud computing", los servidores de Internet se convierten en el sitio principal de almacenaje y ejecución de los programas, enviando una copia o una última actualización a los ordenadores de los usuarios. Así, los sistemas operativos trabajan para coordinarse con una red de aparatos interconectados vía Internet y pasan a estar centrados en la red. En este sentido, los usuarios podrían hacer una fotografía con su teléfono móvil e inmediatamente tenerla disponible en su ordenador de sobremesa.

Como ejemplos de Computación en Nube destacan Amazon Web Ser-

vices (EC2, S3...), Google Apps, eyeOS y Microsoft Azure, que proveen aplicaciones comunes de negocios en línea accesibles desde un navegador web, mientras el software y los datos se almacenan en los servidores propios de cada compañía.

Figura 2.2: Cloud Computing

2.4.4. Polémica

La polémica viene por ciertas corrientes de pensamiento que sostienen que la centralizacion de los servicios en determinados proveedores no será más que un paso atrás a los años 50 o 60, cuando existían terminales "tontos" que lo único que hacían era comunicarse con los servidores para que estos llevasen a cabo una determinada tarea. De volver a esta arquitectura, la información se alojará en los servidores de las distintas compañías, lo que provocará una limitación en la libertad de los usuarios y los hará dependientes del proveedor de servicios, ya que, por ejemplo, no podrán instalar nuevas aplicaciones sin consentimiento de los adminitradores.

Otros, como los seguidores del sofware libre afirman que la computación en nube pone en peligro las libertades de los usuarios, porque éstos dejan su privacidad y datos personales en manos de terceros. Además afirman que la computación en nube es una trampa destinada a obligar a más gente a adquirir sistemas propietarios, bloqueados, que les costarán más y más conforme pase el tiempo.

2.4.5. Amazon EC2

Un producto también de Amazon, en la misma linea que S3 de fomentar los servicios web, es Amazon EC2. EC2 es un pool de servidores públicos que el cliente podrá alquilar para aprovechar su potencia de cálculo. Con EC2, los clientes crean sus propias Imágenes de Máquinas Amazon (AMI Amazon Machine Images) que contienen un sistema operativo, aplicaciones y datos, y controlan cuántas instancias de cada AMI se ejecutan a la vez. El cliente paga por las horas-instancia (y ancho de banda) que usa, agregando recursos en horarios pico y quitándolos cuando no se necesitan más.

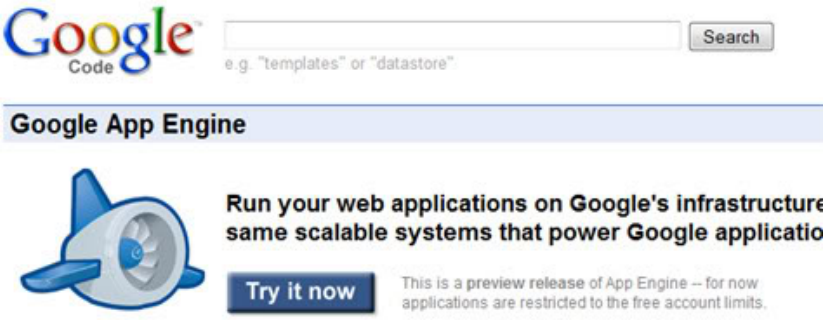
Amazon brinda cinco tipos diferentes de servidores, desde servidores de un solo procesador x86 hasta servidores de ocho procesadores x86_64. No es necesario saber qué servidores se están usando para crear instancias de servicios. Se pueden ubicar instancias en ubicaciones geográficas distintas, o en zonas.

Por la naturaleza de los servicios web, Amazon EC2 puede trabajar conjuntamente con Amazon S3, asi como para otros servicios del mismo Amazon como SimpleBD, servicio de base de datos.

2.4.6. Google App Engine

También Google apuesta por el "cloud computing", con una plataforma denominada Google App Engine. Este servicio ofrece gratuitamente, durante un periodo de prueba, 500 Megabytes de espacio en Internet y la suficiente capacidad de procesado y ancho de banda para servir cinco millones de páginas al mes. Además, los desarrolladores disponen de 10 Gigabits por segundo

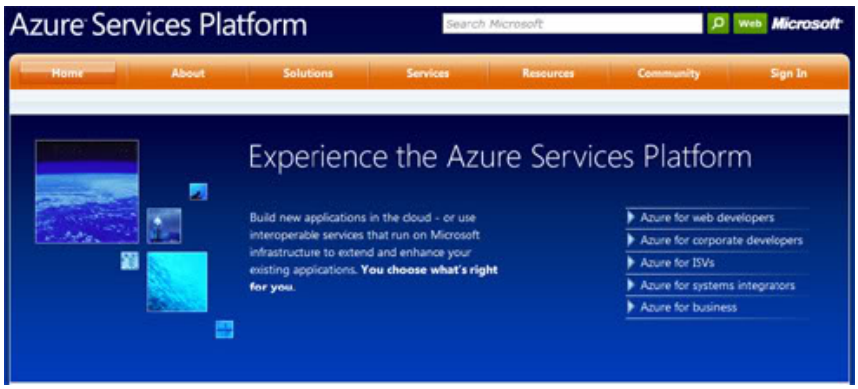
al día de transmisión por cada aplicación.



Google App Engine está basado en la tecnología que utiliza Google para sus propias aplicaciones. Para el sistema de almacenamiento distribuido, dispone de la tecnología Bigtable, y del sistema de ficheros GFS.

2.4.7. Microsoft Azure

Presentado el pasado mes de octubre, Microsoft Azure es la nueva plataforma de Microsoft para el almacenamiento de ficheros, administración de servicios y computación orientado a desarrolladores y empresas. En la primera fase, Azure es sólo compatible con tecnologías propietarias de Microsoft como Live, SQL y .NET Services, Dynamics CRM y SharePoint, aunque pronto también será compatible con otros lenguajes de programación basados en estándares abiertos.



A pesar de no estar disponibles ni precios ni detalles sobre la plataforma Azure más allá de su primera versión de prueba para desarrolladores, se

espera su lanzamiento a principios de 2010, coincidiendo con la reciente llegada del nuevo sistema operativo Windows 7. Esta nueva versión de Windows será una parte fundamental de Azure, ya que permitirá la interoperabilidad entre ambas plataformas.

Microsoft quiere convertir a Azure en una parte central en el desarrollo de aplicaciones para las diferentes plataformas y servicios de la empresa. De esta manera, las empresas y usuarios pueden utilizar servicios y aplicaciones que estén interelacionadas entre ellas en los diferentes dispositivos que utilicen.

Capítulo 3

Diseño

Como se indicó anteriormente, el punto de partida del proyecto es el archivo AmazonS3.wsdl que Amazon ofrece en su pagina web para que los desarrolladores adapten sus aplicaciones a Amazon S3. Se puede consultar esta interfaz en la siguiente direccion:

http://s3.amazonaws.com/doc/2006-03-01/AmazonS3.wsdl

A partir de dicho archivo wsdl, y por medio de la herramienta web que ofrece la pagina oficial de gSOAP, se construyó el archivo de cabeceras wsdl.h. En este archivo, se define la misma interfaz que teniamos en el wsdl, pero en codigo C. Compilando este fichero obtenemos el esqueleto de la aplicación.

A partir del esqueleto generado se comenzó a construir la aplicacion. Se fue desarrollando operación a operación, primero en la parte del servidor y luego, para poder probarlo, en la parte del cliente.

Se barajó la posibilidad de que el cliente fuera interactivo, pero no se consideró útil porque la mayoria de las veces se harán pocas operaciones y no merece la pena arrancar el programa para introducir los comandos. Por ello se decidió que todos los comandos se ejecutaran desde la línea de comandos clásica de cualquier sistema UNIX. Todos ellos llamarán al programa "c3" y dentro del comando llevarán el codigo de la operación que se quiere realizar, así como los parámetros necesarios. El esquema típico de la llamada será:

c3 codigo-operacion [parámetros]

La forma de proceder de la aplicación en todas las operaciones sigue un mismo esquema básico:

1. El cliente realiza la petición y se queda bloqueado esperando la respuesta.

2. El servidor resuelve la petición y devuelve el resultado.

3. El cliente recibe el resultado y, si es necesario, muestra la información por pantalla

Por defecto el servidor estará escuchando en el puerto 8080, pero este se podrá cambiar en los parámetros de la llamada o del fichero de configuración. Igualmente los clientes usarán el mismo puerto por defecto pudiendo cambiarse si la situacion lo requiere para adaptarse al servidor.

3.1. Diseño multihilo en el servidor

El diseño multihilo del servidor permitirá atender varias peticiones a la vez. Esto es una característica fundamental de cualquier servidor, y con las facilidades que nos proporciona gSOAP no supone una gran diferencia con un servidor secuencial en lo que a implementación se refiere.

Cuando el servidor recibe una petición, crea un nuevo proceso con la informacion de dicha petición. Este nuevo proceso es el que lleva a cabo la operación requerida y devuelve el resultado. En la figura 3.1 se puede ver un diagrama de secuencia para esta interacción.

3.2. Gestion de usuarios

En el apartado de gestión de usuarios, la aplicación lleva un registro de cada nombre de usuario junto con su clave privada de cifrado. Esta clave será la usada para el proceso de autenticación. Inicialmente esta información se guarda en un fichero de texto plano, puesto que no se ha sometido a la aplicación a grandes exigencias de almacenamiento, pero en un futuro se podría ampliar con una base de datos si fuese necesario (ver apartado de "Líneas Futuras").

Cada usuario tendrá asignado en el servidor una carpeta UNIX, que representará su espacio de usuario, donde podrá crear buckets y objetos. En un principio no hay límite de espacio, pero en una supuesta versión comercial no estaría de más limitarlo por razones obvias. Así lo hace Amazon S3. Dicha carpeta de usuario llevará como nombre el AWSAccessKeyId del usuario, por

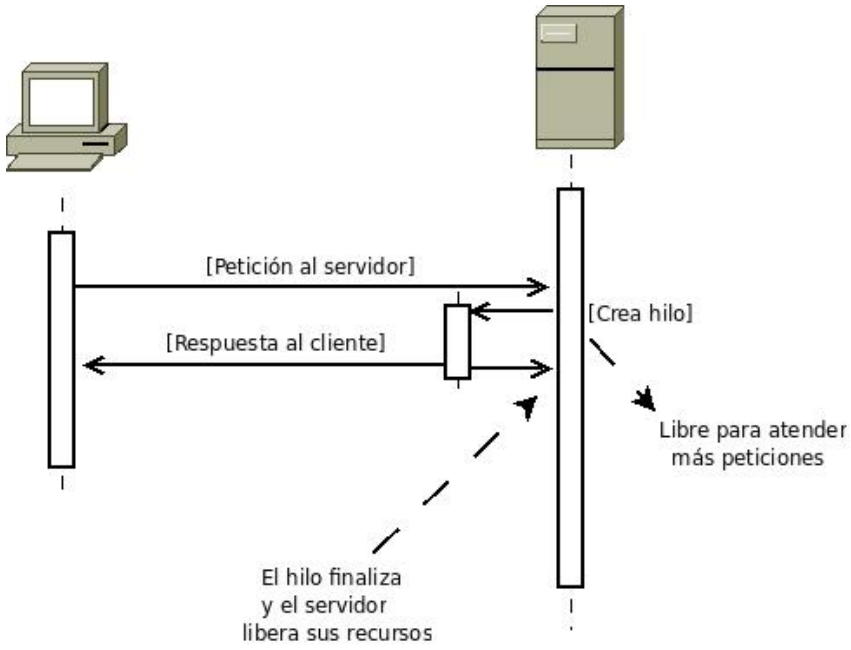


Figura 3.1: Diseño multihilo del servidor

lo que el sistema no soporta dos usuarios con el mismo AWSAccessKeyId. De momento, como el registro de usuarios lo lleva de manera manual el administrador de la aplicación, será su responsabilidad el cumplir este requisito. En un futuro se debería controlar esto.

3.3. Almacenamiento en el servidor

Cada bucket en el servidor será a su vez una carpeta UNIX. Amazon S3 permite la compartición de buckets entre distintos usuarios. En el caso de este proyecto, también se permitirá compartir tanto buckets como objetos. La simulación de esta compartición se llevará a cabo usando links simbólicos, de manera que cuando un usuario comparta un bucket o objeto con otro usuario, se creará un enlace simbólico en el espacio del segundo usuario que apuntará al objeto o bucket original.

Al igual que en Amazon S3 no se permitirá la creación de buckets anidados. La implementación de esta característica no supondría gran dificultad, pero requeriría modificar la interfaz que ofrece Amazon S3. Por ello hemos optado por rechazar esta característica e incluirla en unas posibles lineas futuras.

3.4. Autenticacion de usuarios

Para la autenticación de usuarios se ha implementado el mismo sistema que en Amazon S3. Este sistema consiste en que en cada uno de los mensajes SOAP que el cliente le mande al servidor existirá un campo 'Signature' que contendrá una cadena cifrada con el algoritmo HMAC-SHA1.

Dicho campo 'Signature' almacenará el resultado de cifrar una cadena de la forma que sigue:

'AmazonS3' + OPERACION + Timestamp

Incluyendo el 'Timestamp' nos aseguramos que cada mensaje sea único. Un ejemplo de cadena antes de aplicarle el cifrado sería el siguiente:

AmazonS3CreateBucket2008-03-01T12:00:34.000Z

A continuación, el cliente cifrará dicha cadena con su clave secreta, que solo conocen él y el servidor. Para el cifrado se usará el algoritmo antes comentado HMAC-SHA1. La cadena cifrada se incluirá como parte de cada mensaje.

El servidor lo que hará será el mismo proceso, usando la clave del cliente que guarda almacenada en el fichero de claves. Una vez que, usando el mismo algoritmo de cifrado, obtenga una cadena cifrada, la comparará con la firma que le ha enviado el cliente y en función del resultado autenticará o no al cliente.

En la figura 3.2 se muestra un esquema del proceso de autenticación.

3.5. Arranque y configuración

En esta sección se describe el proceso de arranque del servidor y del cliente.

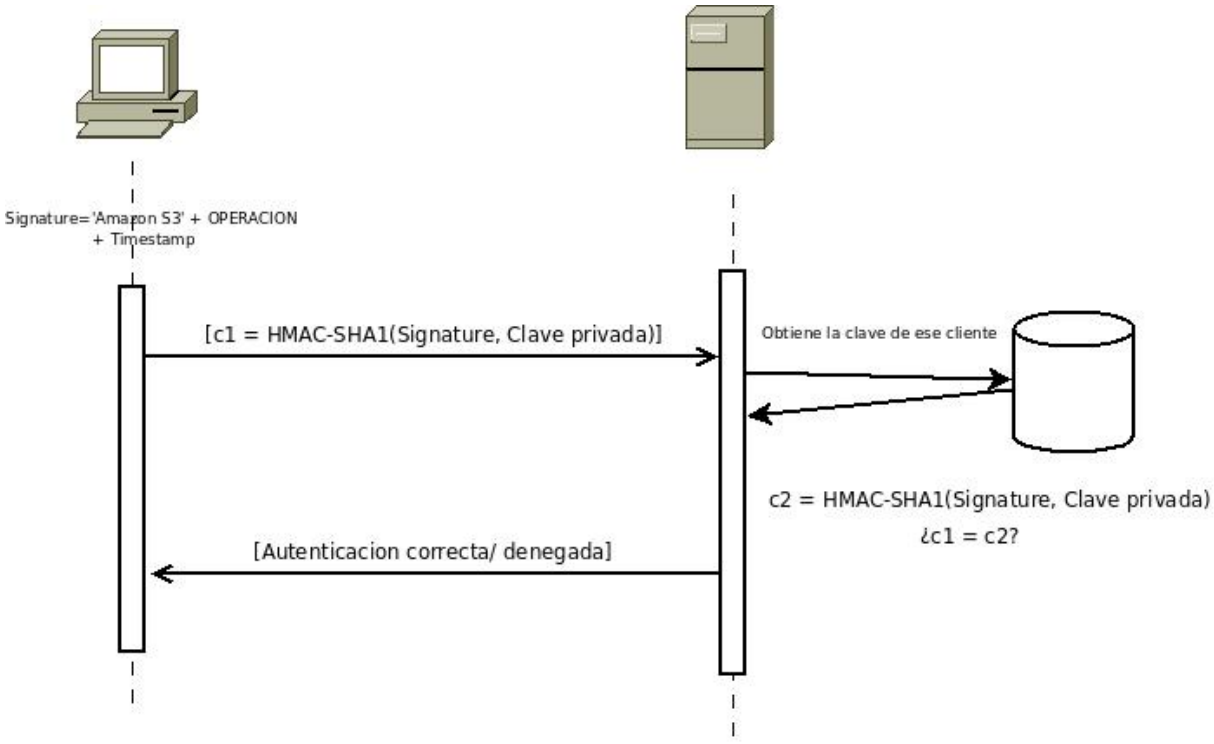


Figura 3.2: Diagrama que muestra el proceso de autenticación

3.5.1. En el servidor

Cuando se arranque el servidor, este lleva acabo una serie de comprobaciones para garantizar el correcto funcionamiento del mismo.

Lo primero que hace el servidor es leer su fichero de configuración. De él extrae los valores que han de tomar las siguientes variables:

- USERS_DIR: representa el directorio donde se almacenarán los buckets de los clientes.
- METADATA_DIR: contiene la ruta del directorio donde se almacenarán los metadatos de los buckets y objetos de los clientes.
- USERS_FILE: contiene la ruta al fichero con la información de acceso de cada uno de los usuarios registrados en la aplicación.

Una vez leído el fichero de configuración, el servidor lee los parámetros que recibe por línea de comandos. Estos parámetros pueden ser dos (ambos son opcionales):

-d: modo depuración activado. Muestra por pantalla las trazas de la ejecución.
-p: va seguido del número de puerto en el que el servidor escuchará peticiones.
Por tanto, el comando para arrancar el servidor será como sigue:

c3_Server [-d] [-p 7686]

Esto arrancará el servidor en el puerto 7686. Por defecto se arrancará en el 8080.

A continuación vemos un ejemplo de fichero de configuración del servidor, que deberá ser un fichero de texto con codificación UTF8:

```
USERS_DIR       ./users/
METADATA_DIR    ./metadata/
USERS_FILE      ./config/bbdd
```

El nombre de la variable y su valor deberán estar separados por un carácter tabulador.

El nombre del fichero por defecto será "c3_server_config" y estará en el directorio "/etc/" de la máquina servidor.

3.5.2. En el cliente

Igual que ocurre con el servidor, el cliente también necesita ciertos parámetros para poder llevar a cabo su labor. Estos parámetros son los siguientes:

1. SERVER: dirección IP del servidor con el que quiere comunicarse.

2. PORT: número de puerto en el que escucha el servidor.

3. ID: clave de identificación del usuario, por tanto es única e identifica unívocamente a cada usuario. Consta de una cadena alfanumérica de 20 caracteres de longitud. En Amazon S3 se llama AWS Access Key Id.

4. SECRET_KEY: clave secreta para el proceso de autenticación. Consta de una cadena alfanumérica de 40 caracteres de longitud.

5. DOWNLOADS: contiene la ruta al directorio donde el usurio desea que se le descarguen sus ficheros.

El nombre de la variable y su valor deberán estar separados por un carácter tabulador.

A continuación vemos un ejemplo de fichero de configuración del cliente, que al igual que el del servidor deberá ser un fichero de texto con codificación UTF8:

```
SERVER      http://localhost
PORT        7686
ID          022QF06E7MXBSH9DHM02
SECRET_KEY  kWcrlUX5JEDGM/LtmEENI/aVmYvHNif5zB+d9+ct
DOWNLOADS   /home/fonti/Documentos
```

El nombre del fichero por defecto será “c3_client_config” y estará en el directorio “/etc/” de la máquina cliente.

Capítulo 4

Implementación en el lado del servidor

A continuación pasaremos a describir cada uno de los servicios que ofrece el servidor. Veremos cómo tiene que ser la petición SOAP que envíe el cliente; esto es básicamente el nombre del servicio y los parámetros que lleva. De este modo, al ser un mensaje SOAP, el servidor podrá interactuar con cualquier cliente que use dicha tecnología, sin importar la plataforma de ambos.

4.1. Operaciones sobre el servicio

Sobre el servicio en general, el servidor solo ofrecerá una operación, que es el listado de todos los buckets del usuario.

4.1.1. Listar todos los buckets

Esta operación muestra el nombre de todos los buckets del usuario, así como la fecha de creación del mismo.

A continuación vemos un ejemplo de peticion SOAP para este servicio.

```
<ListAllMyBuckets xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <AWSAccessKeyId>1D9FVRAYCP1VJEXAMPLE=</AWSAccessKeyId>
  <Timestamp>2006-03-01T12:00:00.183Z</Timestamp>
  <Signature>luyz3d3P0aTou39dzbqaEXAMPLE=</Signature>
</ListAllMyBuckets>
```

donde:

`AWSAccessKeyId`: número de identificación de usuario.

`Timestamp`: fecha de creación del mensaje.

`Signature`: firma para la autenticación del usuario.

El mensaje de respuesta del servidor será del siguiente formato:

```
<ListAllMyBucketsResult xmlns="http://s3.amazonaws.com/doc/2006-03-01">
  <Owner>
    <ID>
      bcaf1ffd86f41caff1a493dc2ad8c2c281e37522a640e161ca5fb16fd081034f
    </ID>
    <DisplayName>webfile</DisplayName>
  </Owner>
  <Buckets>
    <Bucket>
      <Name>quotes;/Name>
      <CreationDate>2006-02-03T16:45:09.000Z</CreationDate>
    </Bucket>
    <Bucket>
      <Name>samples</Name>
      <CreationDate>2006-02-03T16:41:58.000Z</CreationDate>
    </Bucket>
  </Buckets>
</ListAllMyBucketsResult>
```

Dentro de este mensaje el cliente recibe los datos de los buckets asociados a su cuenta.

4.2. Operaciones sobre buckets

En esta seccion veremos en detalle las operaciones que tienen lugar sobre los buckets. Se han implementado cuatro de ellas, dejando a un lado operaciones presentes en Amazon S3 como GetBucketAccessControlPolicy y Get/SetBucketLoggingStatus. No obstante, el servidor admitirá este tipo de peticiones, ya que están presentes en su interfaz, pero devolverá un error cuyo

texto será 'Server.NotImplemented'.

4.2.1. Crear bucket

Esta funcionalidad permite al usuario crear un bucket dentro de su espacio de usuario, lo que le permitirá guardar un numero indefinido de objetos. A continuación se muestra el mensaje SOAP de petición que reconoce el servidor:

```
<CreateBucket xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <Bucket>quotes</Bucket>
  <AWSAccessKeyId>1D9FVRAYCP1VJEXAMPLE=</AWSAccessKeyId>
  <Timestamp>2006-03-01T12:00:00.183Z</Timestamp>
  <Signature>luyz3d3P0aTou39dzbqaEXAMPLE=</Signature>
</CreateBucket>
```

donde:

`Bucket`: nombre del bucket a crear.

`AccessControlList`: lista de control de acceso al bucket, es decir, los usuarios que tienen acceso. No se contempla esta posibilidad así que el servidor ignorara este campo. Para establecer permisos de acceso a otro usuario se debe usar la llamada de "Compartir Bucket".

`AWSAccessKeyId`: número de identificación de usuario.

`Timestamp`: fecha de creación del mensaje.

`Signature`: firma para la autenticación del usuario.

Un ejemplo de respuesta SOAP del servidor será el siguiente:

```
<CreateBucketResponse xmlns="http://s3.amazonaws.com/doc/2006-03-01">
  <CreateBucketResponse>
    <Bucket>quotes</Bucket>
  </CreateBucketResponse>
</CreateBucketResponse>
```

4.2.2. Borrar bucket

Esta operación permite al usuario borrar un bucket. Para que esto sea posible, el bucket deberá estar vacio, es decir, no debe contener ningun objeto. Vemos ahora un ejemplo de peticion SOAP del cliente:

```
<DeleteBucket xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <Bucket>quotes</Bucket>
  <AWSAccessKeyId> 1D9FVRAYCP1VJEXAMPLE=</AWSAccessKeyId>
  <Timestamp>2006-03-01T12:00:00.183Z</Timestamp>
  <Signature>luyz3d3P0aTou39dzbqaEXAMPLE=</Signature>
</DeleteBucket>
```

donde:

`Bucket`: nombre del bucket a borrar.

`AWSAccessKeyId`: número de identificación de usuario.

`Timestamp`: fecha de creación del mensaje.

`Signature`: firma para la autenticación del usuario.

Un ejemplo de respuesta del server será el siguiente:

```
<DeleteBucketResponse xmlns="http://s3.amazonaws.com/doc/2006-03-01">
  <DeleteBucketResponse>
    <Code>204</Code>
    <Description>No Content</Description>
  </DeleteBucketResponse>
</DeleteBucketResponse>
```

Está compuesto por un código y una descripción que hacen referencia al resultado de la ejecución.

4.2.3. Listar bucket

Esta funcionalidad permite obtener el nombre de los objetos de un determinado bucket. Además, se puede especificar en cierta manera el listado con los parámetros que describiremos a continuación tras ver la peticion SOAP:

```
<ListBucket xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <Bucket>quotes</Bucket>
  <Prefix>N</Prefix>
  <Marker>Ned</Marker>
  <MaxKeys>40</MaxKeys>
  <AWSAccessKeyId>1D9FVRAYCP1VJEXAMPLE=</AWSAccessKeyId>
  <Timestamp>2006-03-01T12:00:00.183Z</Timestamp>
  <Signature>luyz3d3P0aTou39dzbqaEXAMPLE=</Signature>
</ListBucket>
```

donde:

`Bucket`: nombre del bucket a listar.

`Prefix`: si esta presente, solo devuelve los objetos que comiencen por la cadena que contiene.

`Marker`: si esta presente, solo devuelve objetos cuyo nombre este alfabéticamente después que la cadena que contiene.

`MaxKeys`: número máximo de objetos listados.

`AWSAccessKeyId`: número de identificación de usuario.

`Timestamp`: fecha de creación del mensaje.

`Signature`: firma para la autenticación del usuario.

Un ejemplo de respuesta para una peticion de listar bucket sería el siguiente:

```
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01">
  <Name>quotes</Name>
  <Prefix>N</Prefix>
  <Marker>Ned</Marker>
  <MaxKeys>40</MaxKeys>
  <IsTruncated>false</IsTruncated>
  <Contents>
    <Key>Nelson</Key>
    <LastModified>2006-01-01T12:00:00.000Z</LastModified>
    <ETag>&amp;quot;828ef3fdfa96f00ad9f27c383fc9ac7f&amp;quot;</ETag>
```

```
11      <Size>5</Size>
12      <StorageClass>STANDARD</StorageClass>
13      <Owner>
14        <ID>
15      bcaf1ffd86f41caff1a493dc2ad8c2c281e37522a640e161ca5fb16fd081034f
16        </ID>
17        <DisplayName>webfile</DisplayName>
18       </Owner>
19    </Contents>
20    <Contents>
21      <Key>Neo</Key>
22      <LastModified>2006-01-01T12:00:00.000Z</LastModified>
23      <ETag>&amp;quot;828ef3fdfa96f00ad9f27c383fc9ac7f&amp;quot;</ETag>
24      <Size>4</Size>
25      <StorageClass>STANDARD</StorageClass>
26       <Owner>
27        <ID>
28      bcaf1ffd86f41caff1a493dc2ad8c2c281e37522a640e161ca5fb16fd081034f
29        </ID>
30        <DisplayName>webfile</DisplayName>
31      </Owner>
32  </Contents>
33 </ListBucketResult>
```

En este caso vemos como se devuelve la información de dos objetos.

4.2.4. Compartir bucket

Con esta funcionalidad, un usuario podra compartir sus buckets con otro usuario con cuenta en el sistema. Para ello solo deberá indicar el nombre del bucket a compartir y el nombre del usuario con quien desea compartir dicho bucket. Vemos ahora un ejemplo de petición para este servicio.

```
1 <SetBucketAccessControlPolicy xmlns="http://doc.s3.amazonaws.com/2006-
2   <Bucket>Musica</Bucket>
3   <AccessControlList>
4     <Grant>
5       <Grantee xsi:type="CanonicalUser">
6         <AWSAccessKeyId>
7       a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be666e9
8         </AWSAccessKeyId>
```

```
9           </Grantee>
10          <Permission>READ</Permission>
11        </Grant>
12    </AccessControlList>
13    <AWSAccessKeyId>1D9FVRAYCP1VJEXAMPLE=</AWSAccessKeyId>
14    <Timestamp>2006-03-01T12:00:00.183Z</Timestamp>
15    <Signature>luyz3d3P0aTou39dzbqaEXAMPLE=</Signature>
16 </SetBucketAccessControlPolicy >
```

donde:

`Bucket`: nombre del bucket a compartir.

`AccessControlList`: contiene los permisos y el AWSAccessKeyId del usuario al que se aplican dichos permisos. Por diseño del mensaje XML podría contener más de un permiso para distintos usuarios, característica que el cliente no explota ya que solo lo hace con un usuario de cada vez.

`AWSAccessKeyId`: número de identificación de usuario.

`Timestamp`: fecha de creación del mensaje.

`Signature`: firma para la autenticación del usuario.

Con esto se compartirá el bucket "Musica" con el usuario cuyo AWSAccessKeyId va especificado en la petición. De momento solo se contemplan permisos de lectura (ver apartado "Líneas futuras").

4.3. Operaciones sobre objetos

En esta sección nos centraremos en las operaciones que podremos realizar sobre los objetos en sí. Al igual que en las operaciones sobre buckets, en este apartado tampoco vamos a hablar de las operaciones sobre los permisos de acceso de los objetos (excepto "Compartir objeto"), ya que no se han implementado. Así mismo, la operaciones presentes en Amazon S3 'Put Object In Line' y 'Get Object Extended' no se han implementado puesto que no supone funcionalidad nueva, si no que son variantes del 'Put Object' y del 'Get object' que sí se han implementado.

4.3.1. Subir objeto

Esta función permitirá subir objetos del cliente al servidor. Los datos del objeto formarán parte del mesanje como adjuntos DIME, por lo que no tiene representacion en el mensaje SOAP. A continuacion se muestra en ejemplo de peticion SOAP:

```
<PutObject xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <Bucket>quotes</Bucket>
  <Key>Nelson</Key>
  <Metadata>
    <Name>Content-Type</Name>
    <Value>text/plain</Value>
  </Metadata>
  <Metadata>
    <Name>family</Name>
    <Value>Muntz</Value>
  </Metadata>
  <ContentLength>5</ContentLength>
  <AccessControlList>
    <Grant>
      <Grantee xsi:type="CanonicalUser">
        <ID>
       a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be666e9
        </ID>
        <DisplayName>chriscustomer</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
    <Grant>
      <Grantee xsi:type="Group">
        <URI>http://acs.amazonaws.com/groups/global/AllUsers<URI>
      </Grantee>
      <Permission>READ</Permission>
    </Grant>
  </AccessControlList>
  <AWSAccessKeyId>1D9FVRAYCP1VJEXAMPLE=</AWSAccessKeyId>
  <Timestamp>2007-05-11T12:00:00.183Z</Timestamp>
  <Signature>luyz3d3P0aTou39dzbqaEXAMPLE=</Signature>
</PutObject>
```

donde:

`Bucket`: nombre del bucket destino donde se guardará el objeto.

`Key`: nombre del objeto en el servidor.

`Metadata`: guarda los metadatos asociados al objeto. Cada uno de estos metadatos estará compuesto por una pareja nombre-valor. Estos datos quedarán asociados al objeto para un hipotético uso posterior.

`ContentLength`: la longitud de los datos en bytes.

`AccessControlList`: elemento opcional que se usa para establecer permisos sobre el objeto al propietario del mismo (solo lectura o lectura/escritura).

`AWSAccessKeyId`: número de identificación de usuario.

`Timestamp`: fecha de creación del mensaje.

`Signature`: firma para la autenticación del usuario.

4.3.2. Copiar objeto

Esta función se usa para copiar un objeto ya presente en el servidor, desde un bucket origen a un bucket destino. El resultado es que el objeto se duplica, estando presente tanto en el bucket origen como en el bucket destino.

Un ejemplo de petición SOAP es el siguiente:

```
1 <CopyObject xmlns="http://bucket_name.s3.amazonaws.com/2006-03-01">
2   <SourceBucket>source_bucket</SourceBucket>
3   <SourceObject>source_object</SourceObject>
4   <DestinationBucket>destination_bucket</DestinationBucket>
5   <DestinationObject>destination_object</DestinationObject>
6   <MetadataDirective>{REPLACE | COPY}</MetadataDirective>
7   <Metadata>
8     <Name>metadata_name</Name>
9     <Value>metadata_value</Value>
```

```
10    </Metadata>
11      ...
12    <AccessControlList>
13      <Grant>
14        <Grantee xsi:type="user_type">
15          <ID>user_id</ID>
16          <DisplayName>display_name</DisplayName>
17        </Grantee>
18        <Permission>permission</Permission>
19      </Grant>
20       ...
21    </AccessControlList>
22    <CopySourceIfMatch>etag</CopySourceIfMatch>
23    <CopySourceIfNoneMatch>etag</CopySourceIfNoneMatch>
24    <CopySourceIfModifiedSince>
25        date_time
26    </CopySourceIfModifiedSince>
27    <CopySourceIfUnmodifiedSince>
28        date_time
29     </CopySourceIfUnmodifiedSince>
30    <AWSAccessKeyId>AWSAccessKeyId</AWSAccessKeyId>
31    <Timestamp>TimeStamp</Timestamp>
32    <Signature>Signature</Signature>
33  </CopyObject>
```

Solo comentaremos los campos que realmente se usan en la aplicación:

`SourceBucket`: nombre del bucket origen.

`SourceObject`: nombre del objeto origen.

`DestinationBucket`: nombre del bucket destino.

`DestinationObject`: nombre del objeto destino. Es opcional. Si no esta presente el nombre del objeto será el mismo que en el origen.

`Metadata`: metadatos asociados al objeto.

`AWSAccessKeyId`: número de identificación de usuario.

`Timestamp`: fecha de creación del mensaje.

`Signature`:firma para la autenticación del usuario.

4.3.3. Descargar objeto

Esta operación permite al cliente descargar un objeto desde el servidor a su máquina.

```
<GetObject xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <Bucket>quotes</Bucket>
  <Key>Nelson</Key>
  <GetMetadata>true</GetMetadata>
  <GetData>true</GetData>
  <InlineData>true</InlineData>
  <AWSAccessKeyId>1D9FVRAYCP1VJEXAMPLE=</AWSAccessKeyId>
  <Timestamp>2006-03-01T12:00:00.183Z</Timestamp>
  <Signature>luyz3d3P0aTou39dzbqaEXAMPLE=</Signature>
</GetObject>
```

Al igual que antes, solo comentaremos los campos usados por la aplicación:

`Bucket`:nombre del bucket donde esta guardado el objeto.

`Key`:nombre del objeto.

`AWSAccessKeyId`:número de identificación de usuario.

`Timestamp`:fecha de creación del mensaje.

`Signature`:firma para la autenticación del usuario.

La respuesta tipo para esta peticion será la siguiente:

```
<GetObjectResponse xmlns="http://s3.amazonaws.com/doc/2006-03-01">
  <GetObjectResponse>
    <Status>
      <Code>200</Code>
      <Description>OK</Description>
    </Status>
```

```
7       <Metadata>
8         <Name>Content-Type</Name>
9         <Value>text/plain</Value>
10      </Metadata>
11      <Metadata>
12        <Name>family</Name>
13        <Value>Muntz</Value>
14      </Metadata>
15      <Data>aGEtaGE=</Data>
16      <LastModified>2006-01-01T12:00:00.000Z</LastModified>
17      <ETag>&amp;quot;828ef3fdfa96f00ad9f27c383fc9ac7f&amp;quot;</ETag>
18    </GetObjectResponse>
19  </GetObjectResponse>
```

Vemos como el objeto, al contrario que en la operacion 'Subir Objeto', está presente dentro del mensaje en la etiqueta "Data" en vez de formar parte de un adjunto DIME.

4.3.4. Borrar objeto

Con esta operación el cliente podrá borrar un objeto en el servidor. La peticion SOAP será de este estilo:

```
1  <DeleteObject xmlns="http://doc.s3.amazonaws.com/2006-03-01">
2    <Bucket>quotes</Bucket>
3    <Key>Nelson</Key>
4    <AWSAccessKeyId> 1D9FVRAYCP1VJEXAMPLE=</AWSAccessKeyId>
5    <Timestamp>2006-03-01T12:00:00.183Z</Timestamp>
6    <Signature>luyz3d3P0aTou39dzbqaEXAMPLE=</Signature>
7  </DeleteObject>
```

donde:

`Bucket`: nombre del bucket en el que se encuentra el objeto.

`Key`: nombre del objeto a borrar.

`AWSAccessKeyId`: número de identificación de usuario.

`Timestamp`: fecha de creación del mensaje.

`Signature`:firma para la autenticación del usuario.

La respuesta es muy simple indicando nada más el estado de la ejecución:

```
<DeleteObjectResponse xmlns="http://s3.amazonaws.com/doc/2006-03-01">
  <DeleteObjectResponse>
    <Code>200</Code>
    <Description>OK</Description>
  </DeleteObjectResponse>
</DeleteObjectResponse>
```

4.3.5. Compartir objeto

Con este comando podremos compartir objetos con otros usuarios con cuenta en el sistema. Para ello solo tendremos que indicar el bucket, el nombre del objeto y el AWSAccessKeyId del usuario en cuestión. Vemos a continuacion el ejemplo de petición para esta operación:

```
<SetObjectAccessControlPolicy xmlns="http://doc.s3.amazonaws.com/2006-
  <Bucket>Musica</Bucket>
  <Key>track01.mp3</Key>
  <AccessControlList>
    <Grant>
      <Grantee xsi:type="CanonicalUser">
        <AWSAccessKeyId>
        a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be666
        </AWSAccessKeyId>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
  </AccessControlList>
  <AWSAccessKeyId>1D9FVRAYCP1VJEXAMPLE=</AWSAccessKeyId>
  <Timestamp>2006-03-01T12:00:00.183Z</Timestamp>
  <Signature>luyz3d3P0aTou39dzbqaEXAMPLE=</Signature>
</SetObjectAccessControlPolicy>
```

donde:

`Bucket`:nombre del bucket a compartir.

`Key`:nombre del objeto a compartir.

`AccessControlList`: contiene los permisos y el AWSAccessKeyId del usuario al que se aplican dichos permisos. Por diseño del mensaje XML podría contener mas de un permiso para distintos usuarios, característica que el cliente no explota ya que solo lo hace con un usuario de cada vez.

`AWSAccessKeyId`: número de identificación de usuario.

`Timestamp`: fecha de creación del mensaje.

`Signature`: firma para la autenticación del usuario.

Con esto se compartirá el objeto "track01.mp3" del bucket "Musica" con el usuario cuyo AWSAccessKeyId va especificado en la petición. De momento solo se contemplan permisos de lectura (ver apartado "Líneas futuras").

4.4. Codigos de error

En Amazon S3 se gestionan los errores por medio de unos códigos de error que el servidor devuelve al cliente informándole de lo ocurrido con su petición. En nuestra aplicación no se implementan todos, pero a continuación vemos una tabla con los posibles códigos de error generados, así como una breve explicación:

`Client.AccessDenied`	El cliente trata de acceder a un objeto al que no tiene permiso.
`Client.BucketAlreadyExists`	El cliente trata de crear un bucket que ya existe.
`Client.BucketNotEmpty`	El cliente intenta borrar un bucket que no está vacio.
`Client.InvalidAccessKeyId`	La clave de acceso del cliente es inválida.
`Client.InvalidBucketName`	El nombre del bucket no es una cadena soportada por el sistema.
`Client.InvalidDigest`	La firma que proporciona el cliente no es válida.
`Client.MissingContentLength`	El servidor no recibe el tamaño del contenido del mensaje.
`Client.NoSuchBucket`	El servidor no encuentra el bucket que el usuario le especifica.
`Client.NoSuchKey`	El servidor no encuentra el objeto que el usuario le especifica.
`Server.InternalError`	Error interno del servidor.
`Server.NotImplemented`	El servidor devuelve este error cuando se trata de llamar a una operación soportada por la interfaz pero no impletada en el servidor.

Capítulo 5

Implementación en el lado del cliente

A continuación se describirá la implementación en el lado del cliente, es decir, la manera en la que el cliente llama a las funciones que ofrece el servidor. Este capítulo no pretende ser un manual de uso, para eso ya está el apéndice 1.

Dividimos los métodos en dos tipos:

1. Atómicos: aquellos que se ejecutan con una única llamada del cliente y una respuesta del servidor. Se corresponden con aquellos implementados en el servidor.
2. No atómicos: aquellos que usan varias llamadas al servidor. Se componen de métodos atómicos.

5.1. Métodos atómicos

La estructura del código de estos métodos sigue siempre los mismos pasos, que son:

1. Inicializar la comunicación SOAP por medio de la sentencia "soap_init(&soap)".
2. Rellenar la estructura en C con los parámetros de la llamada, que se corresponden con la interfaz XML definida en el capítulo "Implementación en el lado del servidor".

3. Obtener la firma que se adjunta en el mensaje usando HMAC-SHA1. Se obtiene de la misma manera que se explicó en el servidor.

4. Realizar la llamada al método remoto usando la estructura SOAP inicializada en el pimer paso. El cliente se queda bloqueado hasta que reciba una respuesta.

5. Procesar la respuesta y mostrar los resultados, si procede.

6. Liberar la memoria ocupada.

Los meétodos atómicos son los siguientes (entre paréntesis el nombre en el código C):

1. Listar todo (list_all_my_buckets)

2. Crear bucket (create_bucket)

3. Borrar bucket (delete_bucket)

4. Listar bucket (list_bucket)

5. Subir objeto (put_object)

6. Copiar objeto (copy_object)

7. Obtener objeto (get_object)

8. Compartir objeto (set_object_ACP)

9. Compartir bucket (set_bucket_ACP)

Hay algunos métodos que presentan particularidades, como son los adjuntos DIME, los metadatos y la impresión de resultados por pantalla. A continuación comentamos dichas particularidades.

5.1.1. Adjuntos DIME

El método "put_object" sube un objeto del cliente al servidor. Dicho objeto forma parte del mensaje como adjunto MIME y para ello usa una serie de métodos que son los siguientes:

```
static void *dime_read_open(struct soap *soap, void *handle,
  const char *id, const char *type, const char *options) {
    //lo llama el servidor para abrir el flujo de datos
    return handle;
}

static void dime_read_close(struct soap *soap, void *handle) {
    //lo llama el servidor para cerrar el flujo de datos
    fclose((FILE*)handle);
}
static size_t dime_read(struct soap *soap, void *handle,
  char *buf, size_t len) {
    //lo llama el servidor para leer el contenido del objeto
    return fread(buf, 1, len, (FILE*)handle);
}
```

Estos métodos son llamados por el servidor cuando el cliente sube un objeto, de manera que es el servidor el que lleva la iniciativa de la transferencia, pidiéndole al cliente los bytes que necesite. El cliente, en la llamada, lo único que hace es indicarle al servidor cuáles son estos métodos.

5.1.2. Metadatos

Los objetos que un cliente sube al servidor pueden contener metadatos. Esto es información relacionada con el objeto que puede ser interesante para algo en concreto. En nuestro proyecto, los metadatos estan formados por parejas da datos nombre-valor. Aunque se permite que el cliente adjunte metadatos con cada objeto, de momento estos no se usan para nada. El servidor únicamente los almacena y los devuelve cuando el cliente se lo pide.

Los métodos que usan estos metadatos son el "put_object" y el "get_object". El primero guarda en el servidor los metadatos y el segundo los recupera junto con el objeto que están transfiriendo.

5.1.3. Muestra de resultados

Algunos de los métodos del cliente, a parte de mostrar el tiempo de ejecución, muestran unos resultados. A continuación se muestra una tabla con el método correspondiente y los resultados que muestra por pantalla en una

supuesta ejecución sin errores.

List all my buckets	Muestra todos los buckets de un usuario así como la fecha de su última modificación
Create bucket	No muestra nada
Delete bucket	No muestra nada
List bucket	Muestra los objetos que contiene dicho bucket, asi como su tamaño y su fecha de modificación (tanto del bucket como de los objetos).
Put object	Muestra el nombre del objeto que se está subiendo. Cuando finaliza la transferencia muestra también la velocidad media.
Copy object	No muestra nada.
Get object	Muestra el nombre del objeto que se está obteniendo. Cuando finaliza la transferencia muestra también la velocidad media.
Search object	Muestra los resultados de la busqueda. Para cada resultado muestra el bucket, el tamaño y el nombre del objeto.
Delete object	No muestra nada.
Put bucket	Para cada objeto muestra el nombre en el momento en que se está subiendo. Cuando finaliza la transferencia de cada objeto muestra también la velocidad media.
Get bucket	Para cada objeto muestra el nombre en el momento en que se está obteniendo. Cuando finaliza la transferencia de cada objeto muestra también la velocidad media.
Move object	No muestra nada.
List all	Muestra los objetos que contiene cada uno de los buckets, así como su tamaño y su fecha de modificación (tanto del bucket como de los objetos).

5.2. Métodos no atómicos

En esta sección describiremos las funcionalidades del cliente que, basándose en la interfaz que ofrece el servidor, suponen un añadido a la funcionalidad;

esto es combinando mas de una operación. Por no modificar la interfaz propuesta por Amazon S3 se decidió incluir esta funcionalidad en el lado del cliente, pero bien podría haber sido implementado en el lado del servidor.

5.2.1. Subir bucket

Esta llamada se usa cuando se quiere subir una carpeta UNIX desde el cliente al servidor. Equivaldrá a la creación de un bucket con el nombre de la carpeta (o cualquier otro especificado por parámetro) más tantas llamadas a 'putObject' como objetos haya dentro de la carpeta.

Si la carpeta contiene a su vez otras carpetas solo se subirán los objetos que esten en su primer nivel. Esto es así por restricciones en el lado del servidor, para no crear buckets anidados.

Se puede ver un esquema de la interacción entre el cliente y el servidor en la figura 5.1.

5.2.2. Bajar bucket

Con esta operación el cliente podrá descargar un bucket al completo del servidor. El procedimiento emplea una llamada a 'listBucket' y tantas llamadas a 'getObject' como resultados haya dado 'listBucket'. Puesto que la función de 'listBucket' tenía mecanismos de filtrado de resultados (ver 'listBucket'), podremos emplear esos mismos mecanismos para bajar determinados archivos.

En la figura 5.2 se muestra el esquema de interacción entre el cliente y el servidor.

5.2.3. Mover objeto

Como se ha explicado anteriormente, la función 'copyObject' duplica un objeto para copiarlo a otra carpeta distinta a la origen. Si combinamos esta llamada con 'deleteObject' obtenemos esta nueva llamada.

El esquema de esta llamada se corresponde con la figura 5.3.

5.2.4. Listar todo

Con esta llamada el cliente podrá obtener un listado completo por buckets de los objetos presentes en el servidor. Para ello se combinarán dos llamadas implementadas en el servidor: 'listAllMyBuckets' y 'listBucket'.

La figura 5.4 recoge este proceso de comunicación entre el cliente y el servidor.

5.2.5. Buscar objeto

Esta llamada servirá para buscar un objeto en el servidor dado su nombre o parte de este. El cliente obtendrá todos los resultados coincidentes y los buckets donde se alojan dichos resultados. Es una función parecida a la anterior pero que ejerce un filtrado en los resultados de acuerdo al nombre del objeto. Este filtrado es el último paso de la operación y se realiza en el cliente. No se incluye diagrama ya que es igual que el anterior (Listar todo).

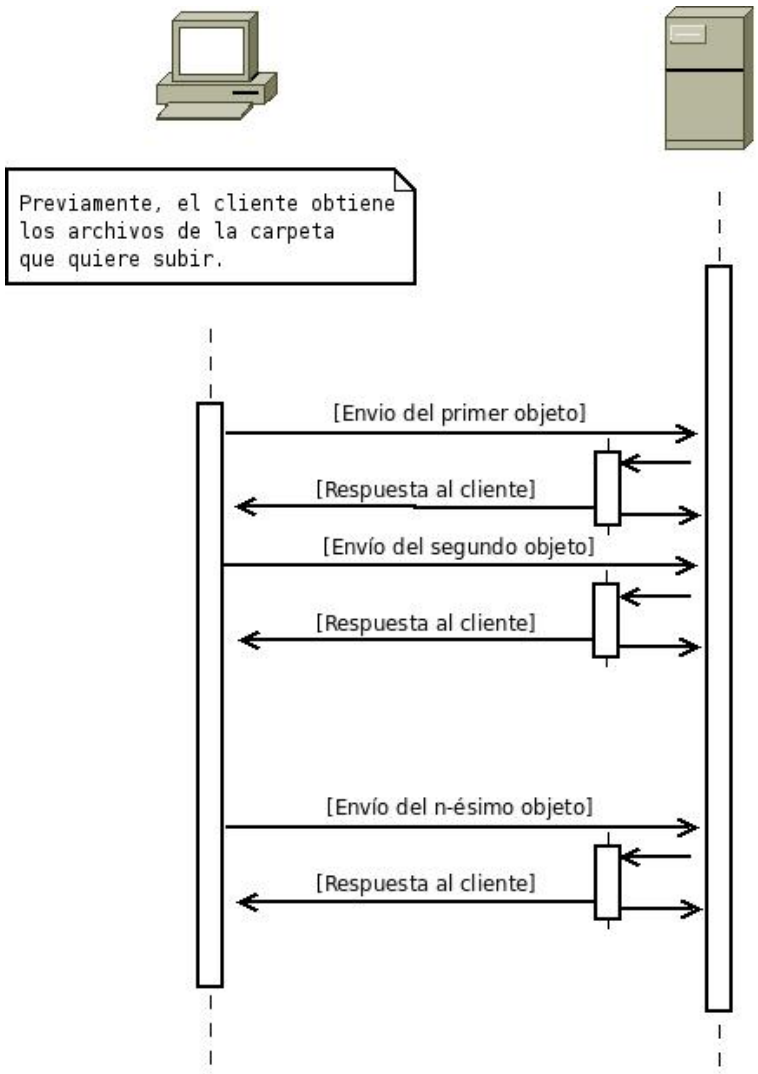


Figura 5.1: Diagrama de la operación de “Subir Bucket”

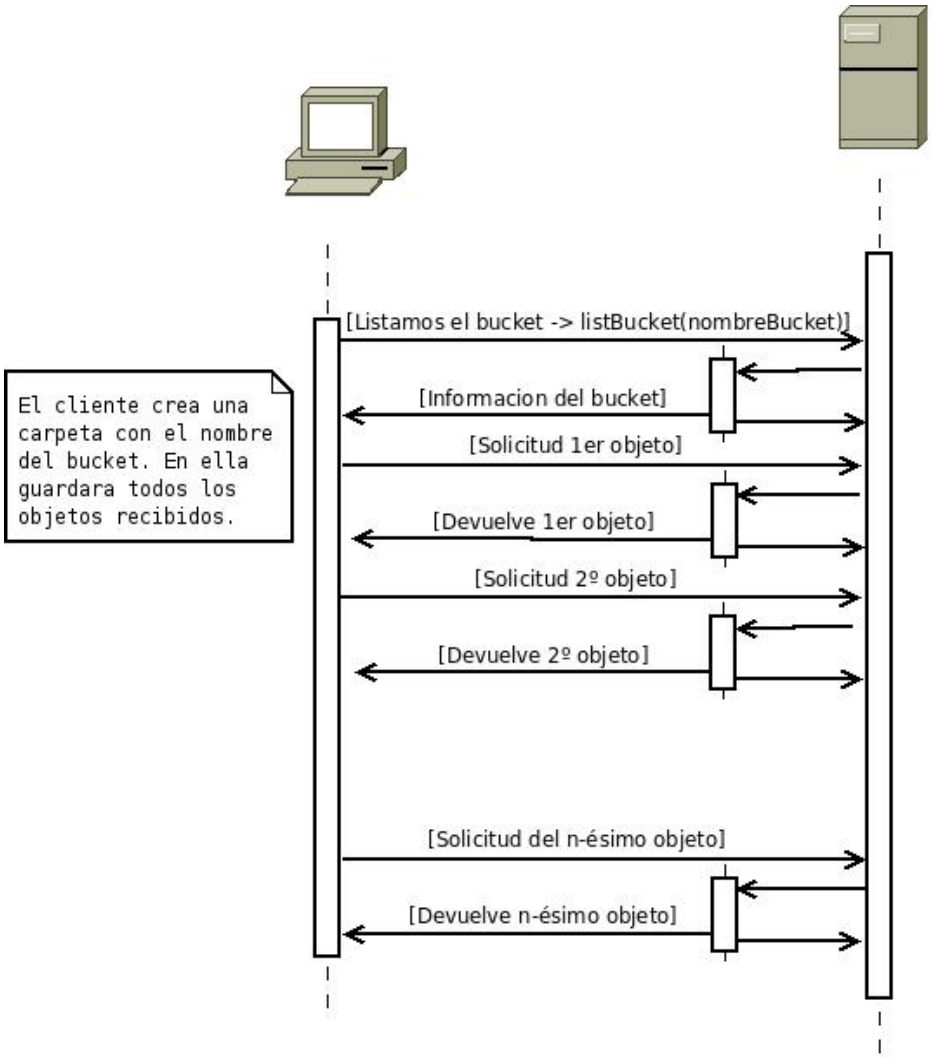


Figura 5.2: Diagrama de la operación de "Bajar Bucket"

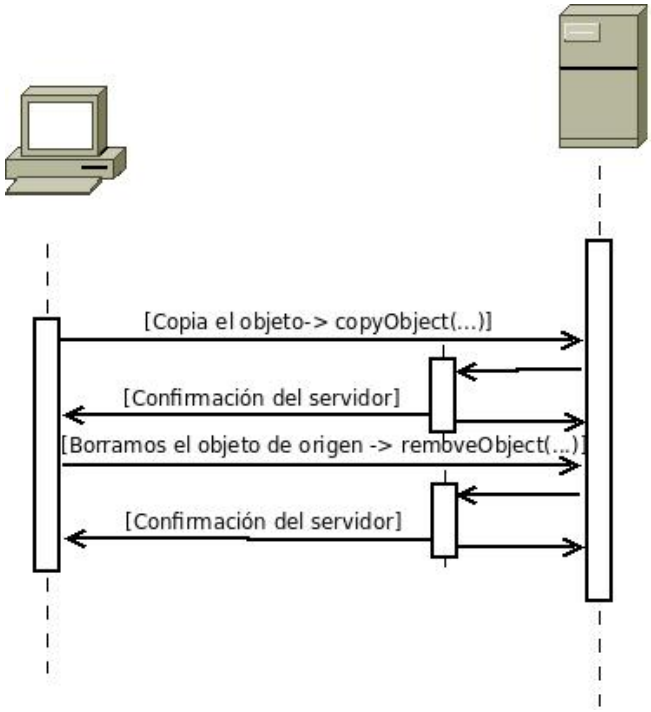


Figura 5.3: Diagrama de la operación de “Mover Objeto”

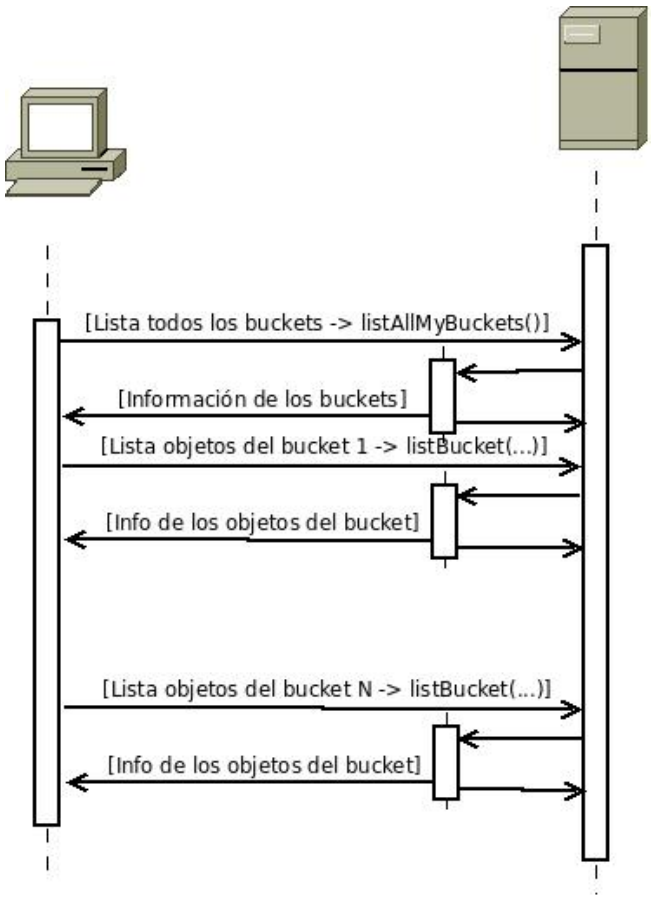


Figura 5.4: Diagrama de la operación de "Listar Todo"

Capítulo 6

Evaluación

En este punto llevaremos a cabo una evaluación de las capacidades del software desarrollado, denominado C3, centrándonos en la velocidad de transferencia, o lo que es lo mismo, el tiempo necesario para transferir archivos de una longitud determinada. Para ello compararemos el sistema con dos de los más importantes protocolos usados para transferencia de ficheros: scp y ftp. Además, para la descarga de contenidos se probará tambien con el programa wget sobre http. Esta alternativa no se puede usar para la subida de archivos.

6.1. Entorno de desarrollo y pruebas

El desarrollo del proyecto, así como las pruebas, se llevaron a cabo sobre Ubuntu 8.04 y 9.04. Para la compilación de los archivos y generación de los stubs se ha usado la version 2.7.9k de las librerías de gSOAP. Esta se puede descargar desde la pagina del proyecto gSOAP (http://www.cs.fsu.edu/ engelen/soap.html).

La versión del compilador de lenguaje C utilizada es la 4.2.4. Es necesario tener instalado la libreria pthread para la compilación y ejecución del servidor.

Para llevar a cabo las pruebas usaremos un ordenador como servidor en la red de la Universidad Carlos III. Para el cliente usaremos un ordenador personal conectado a una línea ADSL de 4Mb (teóricos). Se realizarán pruebas tanto de subida como de bajada para archivos de los siguientes tamaños: 16KB, 64KB, 256KB, 512KB, 1MB, 4MB, 32MB, 64MB y 128MB. A continuación se muestran los resultados obtenidos.

6.2. Subida de archivos

Los resultados han sido muy parecidos para las tres tecnologias probadas. Se ha obtenido una velocidad de subida de unos 50KB/s. En la gráfica se aprecia como apenas hay diferencia en los resultados:

TIEMPOS DE SUBIDA									
TIEMPOS	16KB	64KB	256KB	512KB	1MB	4MB	32MB	64MB	128MB
C3	0,61	1,45	5,06	9,83	19,03	68,35	628,99	1253,87	2460,22
SCP	20,39	21,03	26,45	29,25	36,5	94,41	626,82	1237,11	2432,72
ftp	0,45	1,02	4,31	9,82	18,19	74,88	597,91	1199,01	2439,2

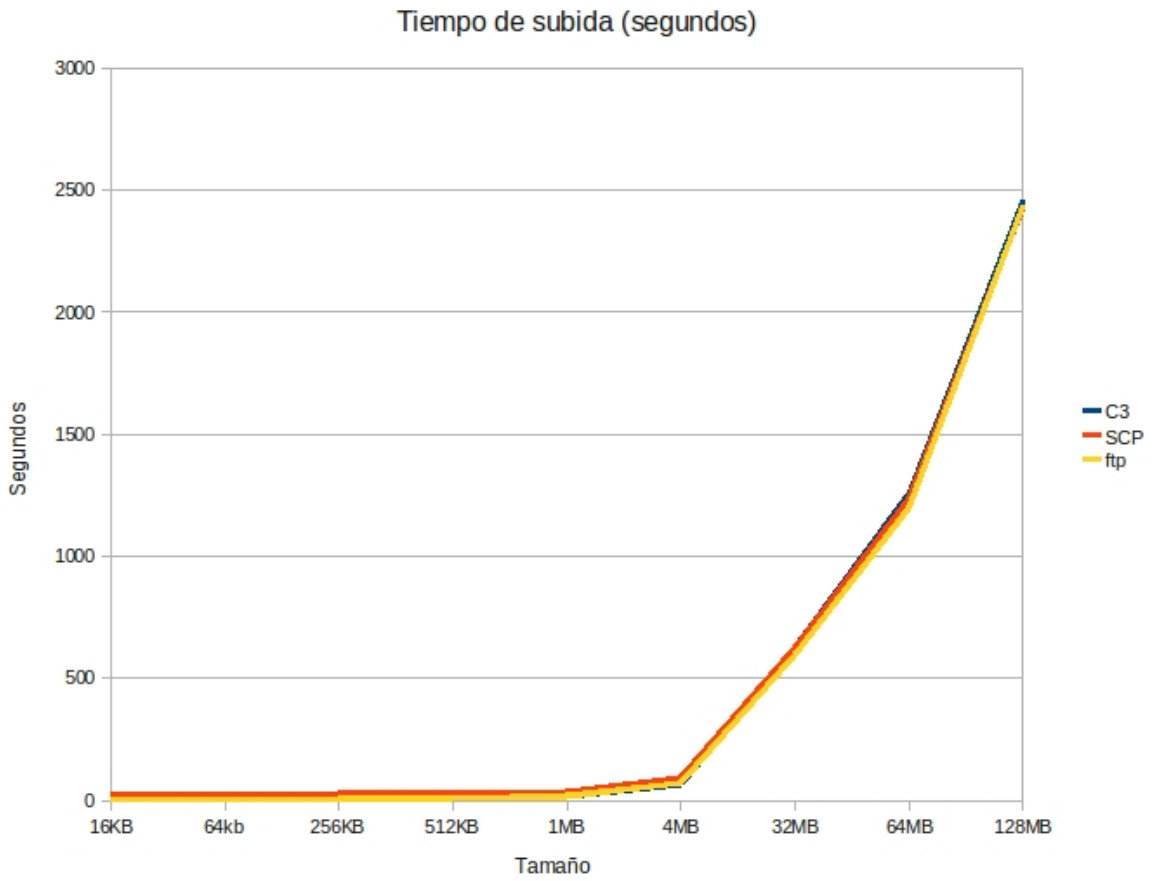


Figura 6.1: Gráfica de los resultados obtenidos en la subida de archivos

6.3. Descarga de archivos

En este caso nuestro sistema C3 se comporta peor que el resto, con unas velocidades que rondan los 300KB/s de media, pero lejos de los 400KB/s de las otras tres tecnologías probadas.

Vemos a continuación una tabla con los resultados medios obtenidos para 5 mediciones y una gráfica para ver claramente los resultados:

TIEMPOS DE BAJADA									
TIEMPOS	16KB	64KB	256KB	512KB	1MB	4MB	32MB	64MB	128MB
C3	0,26	0,44	1,02	1,82	3,38	12,42	96,25	193,46	383,01
SCP	19,2	19,38	19,82	20,48	21,62	26,89	82,12	161,89	313,74
wget/http	0,29	0,39	0,8	1,42	2,55	8,36	69,8	141,21	289,12
ftp	0,23	0,4	0,92	1,51	2,7	9,05	74,87	152,78	300,73

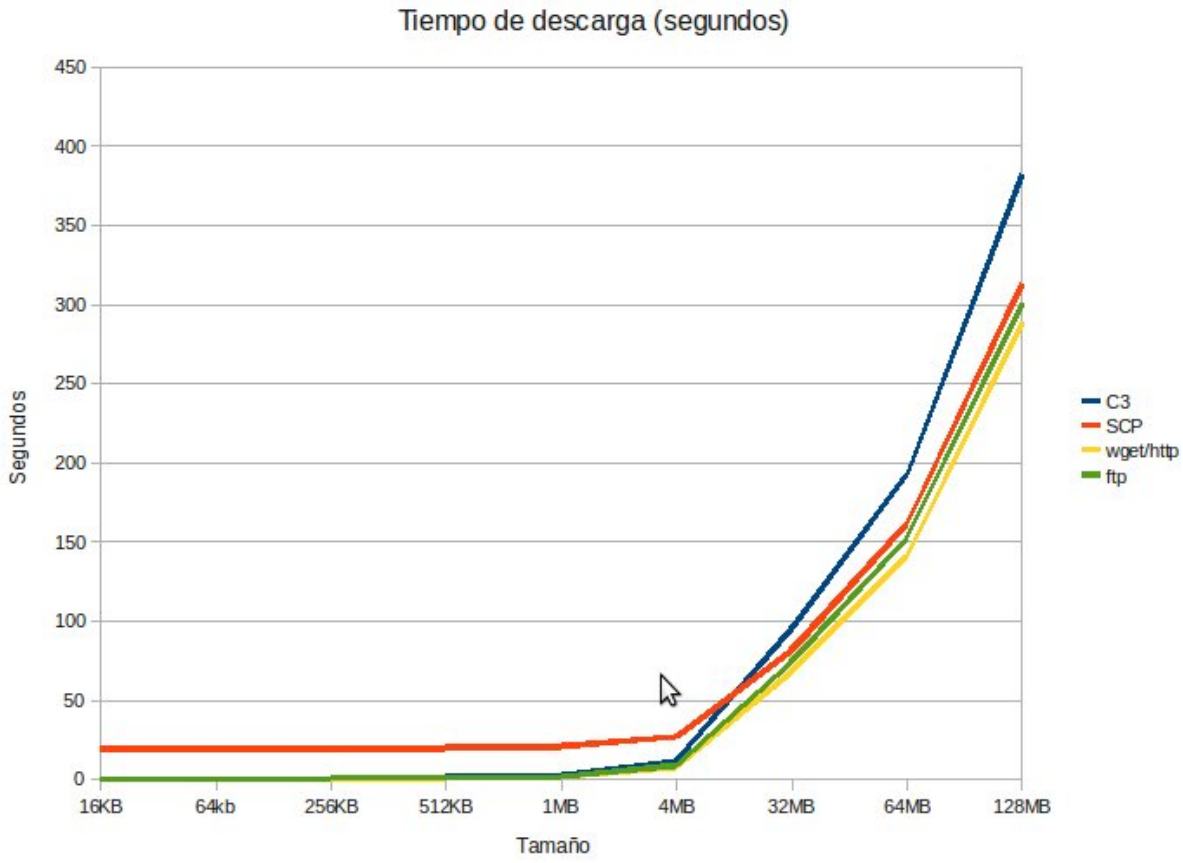


Figura 6.2: Gráfica de los resultados obtenidos en la bajada de archivos

6.4. Analisis de los datos obtenidos

Primero de todo, decir que aunque las gráficas muestren una progresión exponencial, esto no es asi. La progresión es lineal pero la selección del tamaño de los archivos hace que parezca exponencial. Como se puede ver para salir de dudas, los tiempos del archivo de 64 MB son aproximadamente el doble que los del archivo de 32 MB.

Como hemos visto, nuestro sistema se comporta bastante bien en las pruebas de subida, aunque tal vez estén demasiado limitadas por la conexión ADSL en la que se hicieron las pruebas. Los resultados son idénticos comparados con las otras aplicaciones, unas aplicaciones sobradamente contrastadas en el tema de la transferencia de ficheros. En ese aspecto estamos contentos con el resultado obtenido. No es así en los resultados obtenidos en la descarga de archivos, donde nuestro sistema se comporta peor que el resto. Como se indicó anteriormente las velocidades obtenidas con C3 son de unos 300KB/s de media. Con las otras tecnologías se rozaba e incluso sobrepasaba los 400KB/s. Este es el precio que hay que pagar al usar servicios web, ya que existe una sobrecarga introducida por el empaquetamiento de los mensajes XML y su extracción.

Como punto positivo, destacar que C3 es el sistema que menos colapsó la red durante las pruebas. FTP y sobre todo SCP sobreexplotan los recursos dejando sin conexion al resto de aplicaciones de la máquina cliente y ordenadores de la red del mismo.

Cabe destacar el retardo que siempre arrastra SCP desde la primera descarga/subida. Esto se debe al proceso de autenticación en el que el servidor te pide la clave y el cliente debe responder para iniciar la descarga. Aunque se implementó un script para que este proceso de introducir la clave fuese automatico, el retardo en las comunicaciones se ve reflejado en la gráfica. En archivos pequeños SCP se ve penalizado, pero a la larga este inconveniente no se nota tanto. No obstante creemos que este retardo se debe a particularidades del entorno de pruebas, puesto que por experencia, podemos decir que esto no es normal.

Capítulo 7

Conclusiones

Con el desarrollo de este proyecto no hemos adentrado en el mundo de los servicios web, un mundo en plena expansión ya que cada vez son más los desarrollos de software que comtemplan esta tecnología persiguiendo alguna de sus múltiples ventajas.

Al ser una tecnología en plena expansión nos hemos encontrado problemas con la escasa documentación disponible sobre gSOAP. Esta se reducía básicamente a la página web del proyecto gSOAP (http://www.cs.fsu.edu/ engelen/soap.html). En esta web hay una guía donde está toda la teoría referente a gSOAP, pero hemos echado en falta algún ejemplo práctico.

En ese primer momento del desarrollo, cuando apenas teníamos hechas un par de funciones en el servidor, probamos con un cliente de amazon S3 escrito en C# para ver si efectivamente comunicaba con nuestro servidor. Así sucedió. Solo tuvimos que cambiar la dirección del host con el que se comunicaba el cliente para comprobar que la independencia del lenguaje que predica SOAP es una realidad. Este hecho supuso un estímulo para seguir desarrollando el proyecto.

Tras este paso, se decidió implementar paralelamente al servidor un cliente, tambien en C, con el objetivo de probar la funcionalidad del servidor. Hubiese sido más costoso desarrollar un servidor sin un cliente, ya que los clientes de Amazon S3 que encontramos por internet o no tenían toda la funcionalidad que queríamos o estaban hechos en lenguajes que desconocíamos o eran de código cerrado. Ante esto se optó por el desarrollo del cliente.

Trás estas decisiones iniciales, llegaron los problemas con el desarrollo. Los principales los hemos encontrado a la hora de realizar transferencias de

ficheros, y ante la carencia de ejemplos prácticos en la web, no nos ha quedado mas remedio que probar diversas soluciones hasta ver que funcionaba como nosotros queríamos. Ese momento fue un poco frustrante puesto que no conseguíamos transferir archivos, solo mensajes SOAP, y ese, evidentemente, no era el objetivo del proyecto.

Otra decisión importante, una vez que la funcionalidad básica estaba implementada, era el delimitar la extensión del proyecto, puesto que clonar la funcionalidad que ofrece Amazon S3 podría alargar el tiempo de desarrollo de manera alarmante. Como se ha ido indicando a lo largo de esta memoria, las principales funcionalidades de Amazon S3 que han quedado fuera de este proyecto son las siguientes:

- El registro del acceso a bucket y objetos, que permite saber quien ha accedido a que bucket o objeto.
- La trasnferencia de ficheros con el contenido del fichero en el propio mensaje SOAP. En nuestro proyecto se trasnfiere el fichero como adjunto DIME.
- La función "getObjectExtended" que provee mas información a la hora de recuperar un objeto.
- Los permisos han sido reducidos a dos: propietario y no propietario. Cuando un usuario comparte un bucket o objeto con otro usuario, el segundo nunca será propietario por lo que las operaciones que podrá hacer sobre el bucket o objeto serán restringidas.

Una vez desarrollados el cliente y el servidor, la prueba de fuego era hacerlos funcionar en un entorno distinto al local. Normalmente, durante el desarrollo se configuró al cliente para que se conectara a "http://localhost". Una vez que se dispuso de un servidor con IP fija pudimos establecer la comunicación entre ordenadores ubicados en distintas redes. Desde el primer momento, y como era de esperar, el sistema se comportó de la misma manera que lo habia hecho en local, aunque como es lógico las comunciaciones eran más lentas. Esto quizás sea el punto negativo del proyecto, y es que como hemos visto en los resultados C3 pierde con respecto a sus competidores en velocidad de descarga. Pese a ello consideramos los resultados como buenos y creemos que estas carencias se suplen con las ventajas que proporcionan los sevicios web, asi como las ventajas de la propia aplicacion, que permite operaciones útiles para el usuario.

Capítulo 8

Líneas futuras

Varias son las mejoras que se nos han ocurrido a lo largo del desarrollo del proyecto, pero que se ha decidido no incluir; algunas porque alargarían el tiempo de desarrollo de manera excesiva, otras porque no se ajustan a la interfaz propuesta por Amazon y que es el punto de partida de este proyecto. A continuación comentamos las más viables en un futuro.

8.1. Buckets anidados

La organización en carpetas es algo que todos los sistemas operativos de uso común soportan. Además es una metáfora perfectamente entendible por el usuario medio. Esto unido a la facilidad de implementación de dicha característica hacen que sea una mejora a tener en cuenta. En esta versión no se ha tenido en cuenta para ajustarnos a la interfaz de Amazon S3, que recibe el nombre del bucket por parámetro en muchas de las operaciones que permite. El sistema debería permitir una ruta de buckets por parámetros, en vez de un único bucket. Para simular el hecho de tener buckets dentro de buckets, ver el apartado recomendaciones.

8.2. Compartición de buckets

Amazon S3 soporta la compartición de buckets. En este proyecto se soporta una compartición limitada, en la que un usuario podrá compartir un objeto o un bucket con otro usuario pero sin gestión de permisos para los objetos compartidos. Cuando un usuario comparte un objeto con otro, el que recibe el objeto podra descargarlo, moverlo o borrarlo, pero todo esto

será transparente al usuario propietario del objeto, que no verá modificado su objeto.

En Amazon S3 se puede dotar a estas comparticiones de distintos niveles de permisos, pudiendo llegar a tener el mismo control el usuario creador del bucket/objeto como el usuario con el que lo compartió.

8.3. Base de datos

En el entorno de pruebas del proyecto es evidente que no hacia falta una base de datos para gestionar todos los usuarios con sus claves y demás información que pudiera surgir, pero en entornos de producción si podría ser interesante el contar con una.

Además, la inclusión de una base de datos facilitaría la gestión de los usuarios y de los permisos de los buckets que acabamos de ver. Tambien se podrían introducir sesiones en los usuarios, para que un mismo usuario no pueda acceder al mismo objeto a la vez pudiendo provocar una excepción en el sistema. Con un indicador de sesión iniciada se podrían rechazar todas las conexiones de un determinado usuario hasta que finalice la operación que ya tiene iniciada.

En la figura 8.1 se muestra un diagrama de base de datos que podría satisfacer las necesidades de la aplicación.

Con este diseño tan simple de la base de datos conseguimos relacionar los usuarios con sus buckets y a su vez con los objetos de estos. Además nos permite gestionar perfectamente los permisos que tienen tanto sobre los buckets como sobre los objetos. Para ello usamos dos tablas intermedias que relacion usuarios con buckets (User_has_buckets) y buckets con objetos (User_has_objects). Tambien nos hace falta saber quién es el propietario de cada bucket y object, ya que será este el encargado de poder cambiar los permisos. Para ello están los atributos “id_owner” tanto en “Bucket” como en “Object”.

Mediante el atributo “sessionState” de la tabla “User” controlaremos que usuarios tienen una sesión activada en el sistema para evitar accesos inesperados. Otra opción, si no queremos ser tan estrictos y que un usuario pueda hacer varias operaciones al mismo tiempo, es usar el atributo “mutex” de las tablas “Buckets” y “Objects”. Cuando un usuario vaya a modificar cualquier bukect o object bastaría con activar el mutex para que nadie más intente

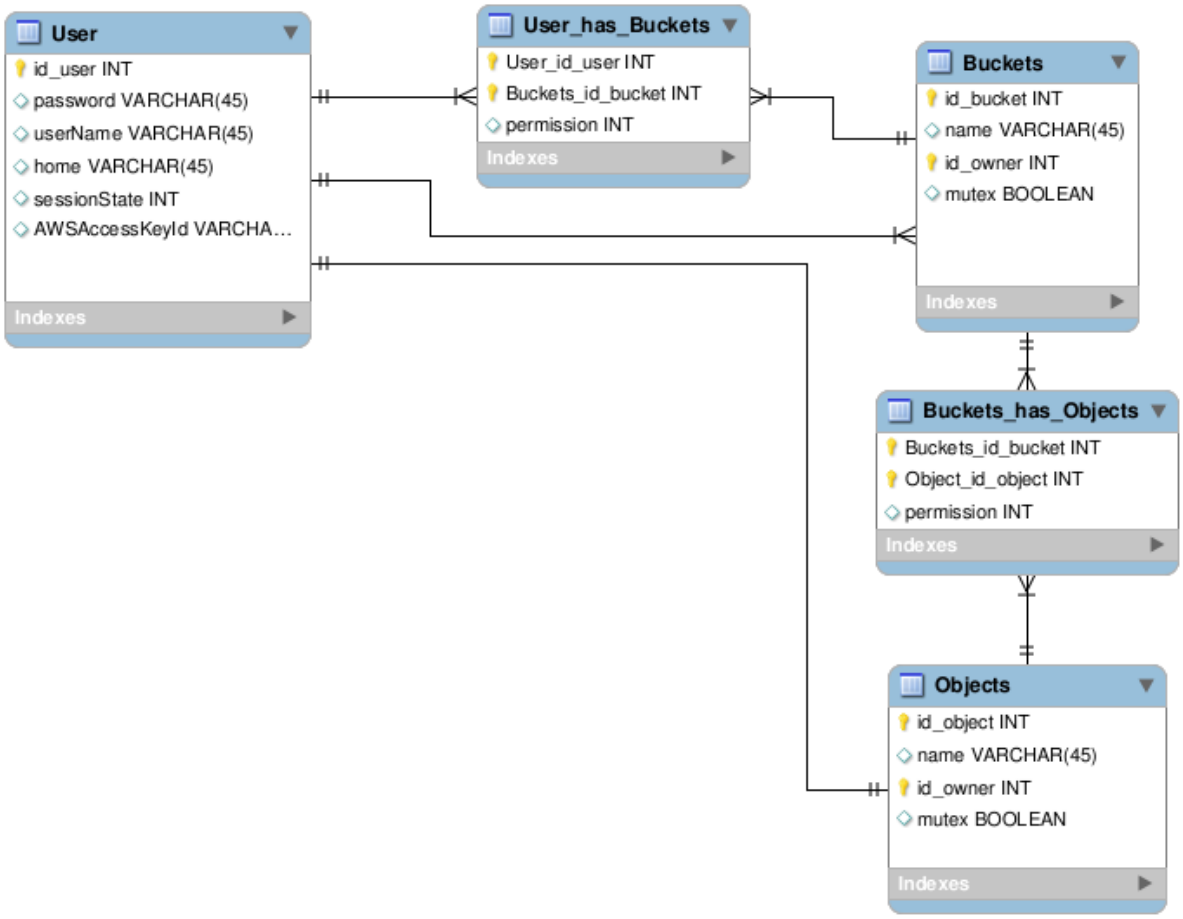


Figura 8.1: Diagrama EER de la base de datos (realizado con MySQL Workbench)

nada sobre dicho bucket o object. Todo esto tiene por objetivo garantizar la exclusión mutua.

Con este diseño de la base de datos y con una interfaz simple de acceso a la misma tendríamos una mejora importante en la aplicación que simplificaría mucho todos los procesos que lleva a cabo el servidor, además de añadir características nuevas.

8.4. Otras operaciones

La inclusion de nuevas operaciones, sobre todo del lado del cliente, aprovechando la interfaz que ofrece el servidor también pueden ser mejoras a tener en cuenta. A continuación vemos una serie de operaciones que se podian implementar sin modificar para nada la interfaz.

8.4.1. Renombrar bucket

Basándonos en la operaciones ya disponibles de create-bucket, move-object y delete-bucket se podría implementar una nueva operacion que renombrara el nombre del bucket. El orden de ejecucion de las operaciones sería:

1. Se crea un bucket con el nombre que le queremos dar.
2. Se mueven todos los archivos del bucket que queremos renombrar al bucket que se acaba de crear.
3. Se borra el bucket con el nombre antiguo.

8.4.2. Operacion para comprobar la velocidad

Una operacion muy sencilla de implementar y a la vez útil sería la de implementar la velocidad de conexión entre el cliente y el servidor, tanto de subida como de bajada.

La operación consistiría en la trasnferencia de un archivo de un tamaño fijo conocido entre cliente y servidor y medir el tiempo que tarda en ejecutarse, algo que ya se hace. Con ello podríamos hallar la velocidad.

Capítulo 9

Presupuesto

En el siguiente apartado se va a mostrar el presupuesto requerido para el proyecto, incluyendo todos los gastos necesarios para su desarrollo y puesta en marcha. El tiempo total para la elaboración del mismo ha sido de 9 meses, desde la concepción inicial de lo que se quería hacer, hasta el momento de elaborar este apartado.

9.1. Fases del proyecto

La siguiente tabla detalla el número aproximado de horas necesarias en cada una de las fases del proyecto.

- Análisis y diseño. Esta fase abarca desde el primer recurso teórico objeto de estudio hasta el momento en que ya se tenga una concepción clara de cómo se va a desarrollar. (250 horas)
- Implementación. El desarrollo puro y duro del código de la aplicación junto con la resolución de los problemas encontrados. (200 horas)
- Pruebas. Fase de pruebas de la aplicación. (150 horas)
- Documentación. Elaboración de una memoria y manual de uso. (200 horas)

9.2. Salarios por categoría

En la elaboracíon del proyecto se requiere personal informático cualificado, el cual deberá adoptar roles distintos relacionados con el tipo de actividades que requiere la elaboración del proyecto. La siguiente tabla muestra el coste de cada rol utilizado:

Cargo	Sueldo neto	Sueldo bruto	Coste/hora
Analista	2.793,6	50.400	45
Documentador	2.483,2	44.800	40
Programador	2.172,8	39.200	35
Responsable de pruebas	2.483,2	44.800	40

- Coste/hora indica el coste para el cliente de una hora de trabajo.
- Sueldo bruto indica el sueldo bruto anual, con 14 pagas mensuales.
- Sueldo neto indica el sueldo neto mensual. Para calcularlo descontamos al sueldo bruto el I.R.P.F (20 %) y la seguridad social (2,4 %).

Se tiene en cuenta que:

- La jornada laboral es de 8 horas.
- Son 20 los días laborables al mes.

9.3. Gastos de personal imputables al proyecto

La siguiente tabla muestra los costes de cada fase del proyecto en función del sueldo de los distintos roles que intervinieron en el proyecto:

Cargo	**Tiempo (horas)**	**Coste/Hora (Euros)**	**Total**
Analista	250	45	11.250
Documentador	200	40	8.000
Programador	200	35	7.000
Responsable de pruebas	150	40	6.000
Total			**32.250**

9.4. Gastos indirectos

La siguiente tabla muestra los gastos indirectos que repercuten en los costes totales del proyecto:

Descripción	**Coste**
Productos de limpieza	Incluido en los costes indirectos
Electricidad	Incluido en los costes indirectos
Agua	Incluido en los costes indirectos
Gastos de comunidad	Incluido en los costes indirectos
Costes indirectos (10 %)	32.250 * 0,10 = 3.225

9.5. Resumen

A continuación se incluye un resumen del presupuesto en el que se incluye el margen de imprevistos y el de beneficio. El total es el coste total del proyecto.

Recurso	**Coste**
Personal con cargo al proyecto	32.250
Gastos indirectos	3.225
Subtotal	35.475
Margen de imprevistos (10 %)	3.547
Total (IVA incluido)	**39.022**

Capítulo 10

Bibliografía

- Amazon. Web de Amazon S3 (http://aws.amazon.com/s3/). Apartados de “Resources”
- Página del proyecto gSOAP: http://www.cs.fsu.edu/ engelen/soap.html
- Apuntes de la asignatura de “Sistemas Distribuidos” de la carrera de Informática (Universidad Carlos III de Madrid)
- Cloud Application Architectures, 1st Edition - Chapter 2 . Amazon Cloud Computing
- Java Web Services Architecture. James McGovern, Sameer Tyagi, Michael Stevens, Sunil Mathew. Chapter 1. Web Services Overview
- Wikipedia (www.wikipedia.org) Se consultaron diversos artículos de Servicios Web, Sistemas distribuidos, wget, SOAP, WSDL...

Capítulo 11

Bibliotecas externas

Para la realización del proyecto, a parte de las bibliotecas gSOAP, se han usado otras dos bibliotecas de apoyo:

"Crypto-avr-lib/microcrypt-lib", desarrollada por Daniel Otte. Esta biblioteca se usó para la generación de la firma. Concretamente se usó el archivo hmac-sha1.c, que contiene una función para cifrar con el algoritmo HMAC-SHA1, que es el empleado por Amazon S3.

"Syncftp: an automatic synchronsing ftp client", desarrollada por Simon Howard y Jon Dowland. De aquí se ha usado el archivo dir.c, que contiene funciones para la lectura de directorios.

Capítulo 12

Apéndice I - Manual de usuario del cliente C3

En este apendice se mostrará al lector cómo usar todas las funcionalidades del cliente, como instalarlo y configurarlo y además una serie de recomendaciones para aprovechar al máximo sus posibilidades.

12.1. Comandos

Dicho cliente se manejará con una serie de comandos desde la consola de UNIX. Estos comandos le permitirán realizar cada una de las operaciones explicadas a lo largo de la memoria del proyecto. Como vimos, la lógica de la mayoría de las operaciones recae sobre el servidor, pero algunas operaciones se basan en varias llamadas, por lo que parte de la lógica la realizada el cliente. Como esto pretende ser un manual de uso, no entraremos en detalle en ese aspecto, simplemente nos limitaremos a explicar cada uno de los comandos con sus respectivos parámetros, ya sean opcionales u obligatorios.

Todos los comandos que invoque el cliente deberán empezar por 'c3' seguido del identificador de la operacion. Cada operacion tiene dos identificadores por los que puede ser llamada indistintamente: el nombre completo de la operación separado por guiones y la abreviatura cogiendo las primeras letras de cada palabra que forma el nombre de la operacion. Se verá mas claro con los ejemplos.

Además de esto, algunos comandos podrán llevar parámetros opcionales.

12.1.1. Listar todos los buckets

Esta función lista el nombre de todos los buckets que el usuario tiene definidos en el servidor. La sintaxis de dicho comando es muy simple y es como sigue:

```
c3 list-all-my-buckets
c3 lamb
```

No tiene parametros.

Este comando muestra por pantalla el listado con el nombre de todos los buckets, además de la fecha de la ultima modificación en de cada uno de ellos. A la izquierda la fecha y a la derecha el nombre del bucket:

```
Wed Mar 4 21:47:50 2009      fotos
Sat Jan 31 21:11:20 2009     videos
Fri Jan 23 17:29:49 2009     archivos
Sun Feb 1 21:02:32 2009      musica
Wed Mar 4 21:48:18 2009      proyecto
```

12.1.2. Crear bucket

Esta función crea un bucket en el servidor. La sintaxis de dicho comando es:

```
c3 create-bucket -b bucket_name
c3 cb -b bucket_name
```

donde:

-b va seguido del nombre del bucket a crear.

Ese comando no devuelve nada si el proceso se completa correctamente.

12.1.3. Borrar bucket

Esta función borra un bucket en el servidor. La sintaxis de dicho comando es:

c3 delete-bucket -b bucket_name
c3 db -b bucket_name

donde:

-b va seguido del nombre del bucket a borrar.

Ese comando no devuelve nada si el proceso se completa correctamente.

12.1.4. Listar bucket

Este comando lista el contenido de un bucket. La sintaxis de dicho comando es:

c3 list-bucket -b bucket_name [-px prefix] [-mk marker] [-max max_keys]
c3 lb -b bucket_name [-px prefix] [-mk marker] [-max max_keys]

donde:

-b va seguido del nombre del bucket a listar.
-px va seguido de una cadena que indica que solo se listarán aquellos objetos que empiecen por dicha cadena.
-mk va seguido de una cadena que indica que solo se listarán aquellos objetos que alfabeticamente vayan por detrás de dicha cadena.
-max va seguido de un número que indica el máximo de objetos listados.

Vamos ahora varios ejemplos para entender los distintos parametros.

Ejemplo 1. Listado completo.

fonti@fonti-desktop: c3 lb -b videos

Fri Jan 23 17:29:47 2009	–> videos/	
Fri Jan 23 17:29:36 2009	692MB	American Gangster.avi
Fri Jan 23 17:29:37 2009	692MB	American History X.avi
Fri Jan 23 17:29:37 2009	683MB	Babel.avi
Fri Jan 23 17:29:38 2009	629MB	Cadena Perpetua.avi
Fri Jan 23 17:29:38 2009	673MB	Ciudad de Dios.avi
Fri Jan 23 17:29:39 2009	723MB	El Pianista.avi

```
Fri Jan 23 17:29:40 2009    654MB    El Ultimo Mohicano.avi
Fri Jan 23 17:29:41 2009    674MB    Enemigo a las Puertas.avi
Fri Jan 23 17:29:41 2009    732MB    La Gran Evasion.avi
Fri Jan 23 17:29:41 2009    592MB    La muerte tenia un precio.avi
Fri Jan 23 17:29:42 2009    582MB    La vida es bella.avi
Fri Jan 23 17:29:42 2009    793MB    Los intocables de Eliot Ness.avi
Fri Jan 23 17:29:43 2009    663MB    Memento.avi
Fri Jan 23 17:29:44 2009    574MB    Mystic River.avi
Fri Jan 23 17:29:45 2009    764MB    Pulp Fiction.avi
Fri Jan 23 17:29:45 2009    642MB    Seven.avi
Fri Jan 23 17:29:46 2009    762MB    Sin perdon.avi
Fri Jan 23 17:29:46 2009    562MB    Snatch.avi
Fri Jan 23 17:29:47 2009    762MB    Wall-e.avi
```

Ejemplo 2. Listado por prefijo

```
fonti@fonti-desktop: c3 lb -b videos -px La
```

```
Fri Jan 23 17:29:41 2009    732MB    La Gran Evasion.avi
Fri Jan 23 17:29:41 2009    592MB    La muerte tenia un precio.avi
Fri Jan 23 17:29:42 2009    582MB    La vida es bella.avi
```

Ejemplo 3. Listado usando marker

```
fonti@fonti-desktop: c3 lb -b videos -mk La
```

```
Fri Jan 23 17:29:41 2009    732MB    La Gran Evasion.avi
Fri Jan 23 17:29:41 2009    592MB    La muerte tenia un precio.avi
Fri Jan 23 17:29:42 2009    582MB    La vida es bella.avi
Fri Jan 23 17:29:42 2009    793MB    Los intocables de Eliot Ness.avi
Fri Jan 23 17:29:43 2009    663MB    Memento.avi
Fri Jan 23 17:29:44 2009    574MB    Mystic River.avi
Fri Jan 23 17:29:45 2009    764MB    Pulp Fiction.avi
Fri Jan 23 17:29:45 2009    642MB    Seven.avi
Fri Jan 23 17:29:46 2009    762MB    Sin perdon.avi
Fri Jan 23 17:29:46 2009    562MB    Snatch.avi
Fri Jan 23 17:29:47 2009    762MB    Wall-e.avi
```

Ejemplo 4. Listado limitando el numero de resultados

fonti@fonti-desktop: c3 lb -b videos -max 7

Fri Jan 23 17:29:36 2009	692MB	American Gangster.avi
Fri Jan 23 17:29:37 2009	692MB	American History X.avi
Fri Jan 23 17:29:37 2009	683MB	Babel.avi
Fri Jan 23 17:29:38 2009	629MB	Cadena Perpetua.avi
Fri Jan 23 17:29:38 2009	673MB	Ciudad de Dios.avi
Fri Jan 23 17:29:39 2009	723MB	El Pianista.avi
Fri Jan 23 17:29:40 2009	654MB	El Ultimo Mohicano.avi

12.1.5. Subir objeto

Esta función se usa para transferir un objeto del cliente al servidor. La sintaxis de dicho comando es:

c3 put-object -b bucket_name -f file [-k key]
c3 po -b bucket_name -f file [-k key]

donde:

-b va seguido del nombre del bucket al que se quiere subir el objeto.
-f va seguido de la ruta local donde se encuentra el objeto.
-k va seguido del nombre que se le da al objeto en el servidor. Si no esta presente, se usará el mismo nombre que tiene en el cliente.

Este comando no devuelve nada si el proceso se completa correctamente.

12.1.6. Copiar objeto

Esta función se usa para copiar un objeto dentro del servidor desde un bucket origen a un bucket destino. La sintaxis del comando es:

c3 copy-object -bsrc source_bucket -nsrc source_name -bdest dest_bucket [-ndest dest_name]
c3 co -bsrc source_bucket -nsrc source_name -bdest dest_bucket [-ndest dest_name]

donde:

-bsrc va seguido del nombre del bucket de origen.
-nsrc va seguido del nombre del objeto de origen.
-bdest va seguido del nombre del bucket de destino.
-ndest va seguido del nombre que se le da al objeto en el destino. Si no se especifica el objeto toma el mismo nombre que tenia en el bucket origen.

Este comando no devuelve nada si el proceso se completa correctamente.

12.1.7. Descargar objeto

Esta función se usa para descargar un objeto presente en el servidor a la maquina del cliente. La sintaxis de este comando es:

c3 get-object -b bucket_name -k key [-dest path]
c3 go -b bucket_name -k key [-dest path]

donde:

-b va seguido del nombre del bucket donde esta almacenado el objeto.
-k va seguido del nombre del objeto.
-dest especifica la ruta donde se almacenará el objeto en el cliente. Si no se especifica se tomará la ruta por defecto establecida en el fichero de configuración del cliente.

Este comando no devuelve nada si el proceso se completa correctamente.

12.1.8. Buscar objeto

Esta función busca un objeto en el servidor dado su nombre o parte de este. La sintaxis del comando es la siguiente:

c3 search-object -k key
c3 so -k key

donde:

-k va seguido del nombre o parte del nombre del objeto que se quiere buscar.

Esta llamada devuelve una lista con los objetos que se ajusta a la busqueda, el tamaño de cada objeto y el bucket que los contiene. A continuación se muestra un ejemplo:

fonti@fonti-desktop: c3 so -k American

videos/ –> 692MB American Gangster.avi
videos/ –> 692MB American History X.avi
musica/ –> 4MB American Idiot.mp3
archivos/ –> 546KB Universidades Americanas.doc

12.1.9. Borrar objeto

Esta función borra un objeto en el servidor. La sintaxis del comando es la siguiente:

c3 delete-object -b bucket_name -k key
c3 do -b bucket_name -k key

donde:

-b va seguido del nombre del bucket donde se encuentra el objeto a borrar.
-k va seguido del nombre del objeto a borrar.

Este comando no devuelve nada si el proceso se completa correctamente.

12.1.10. Subir bucket

Esta función sube el contenido de una carpeta UNIX en el cliente al servidor. Si dicha carpeta contiene a su vez mas carpetas no las subirá, con el fin de mantener la consistencia del servidor y que no existan buckets anidados. Por tanto solo subirá los objetos presentes en el primer nivel de la carpeta. La sintaxis del comando es la siguiente:

c3 put-bucket -f folder_path [-b bucket_name]
c3 pb -f folder_path [-b bucket_name]

donde:

-f va seguido de la ruta, absoluta o relativa, a la carpeta que se quiere subir.

-b va seguido del nombre que se le quiere dar al bucket. Si no se especifica, el nombre será el mismo que el de la carpeta.

12.1.11. Descargar bucket

Esta función descarga el contenido de un bucket desde el servidor al cliente. Se puede filtrar el contenido del bucket a descargar usando los mismos parámetros que en la funcion list-bucket. La sintaxis de este comando es la siguiente:

c3 get-bucket -b bucket_name [-px prefix] [-mk marker] [-max max_keys]
c3 gb -b bucket_name [-px prefix] [-mk marker] [-max max_keys]

donde:

-b va seguido del nombre del bucket a descargar.

-px va seguido de una cadena que indica que solo se descargarán aquellos objetos que empiecen por dicha cadena.

-mk va seguido de una cadena que indica que solo se descargarán aquellos objetos que alfabeticamente vayan por detrás de dicha cadena.

-max va seguido de un número que indica el máximo de objetos descargados.

12.1.12. Compartir objeto

Con esta funcionalidad un usuario podra compartir un determinado objeto con el usuario que introduzca por parámetro. Dicho usuario vera como le aparece un nuevo objeto en su carpeta "share". Este objeto lo podrá leer y modificar sin que esto afecte al usuario que en su momento se lo compartió. La sintaxis de este comando es la siguiente:

c3 share-object -b bucket_name -k key -u user_name
c3 sho -b bucket_name -k key -u user_name

donde:

-b va seguido del nombre del bucket donde se encuentra el objeto a compartir.

-k va seguido del nombre del objeto a compartir

-u va seguido del AWSAccessKeyId del usuario con el que compartir el objeto.

12.1.13. Compartir bucket

Con esta funcionalidad un usuario podra compartir un determinado bucket con el usuario que introduzca por parámetro. Dicho usuario vera como le aparece un nuevo bucket en su espacio de usuario. Este bucket lo podrá leer y modificar sin que esto afecte al usuario que en su momento se lo compartió. La sintaxis de este comando es la siguiente:

c3 share-bucket -b bucket_name -u user_name
c3 sho -b bucket_name -u user_name

donde:

-b va seguido del nombre del bucket donde se encuentra el objeto a compartir.

-u va seguido del AWSAccessKeyId del usuario con el que compartir el objeto.

12.1.14. Mover objeto

Esta función es similar a la función copiar, pero elimina el objeto de su origen. La sintaxis de este comando es la siguiente:

c3 move-object -bsrc source_bucket -nsrc source_name -bdest dest_bucket [-ndest dest_name]
c3 mo -bsrc source_bucket -nsrc source_name -bdest dest_bucket [-ndest dest_name]

donde:

-bsrc va seguido del nombre del bucket de origen.
-nsrc va seguido del nombre del objeto de origen.
-bdest va seguido del nombre del bucket de destino.

-ndest va seguido del nombre que se le da al objeto en el destino. Si no se especifica el objeto toma el mismo nombre que tenia en el bucket origen.

12.1.15. Listar todo

Esta función lista el contenido de todos los buckets, es decir, muestra todos los objetos clasificados por bucket. La sintaxis de este comando es la siguiente:

c3 list-all
c3 la

No lleva parámetros.

Esta función devuelve un listado con el siguiente formato:

```
Sat Jan 31 17:27:17 2009     -> musica/
Sat Jan 31 17:23:51 2009     6MB     01 - Ahora que....mp3
Sat Jan 31 17:24:07 2009     4MB     02 - 19 Días y 500 Noches.mp3
Sat Jan 31 17:24:33 2009     6MB     03 - Barbi Superestar.mp3
Sat Jan 31 17:24:47 2009    3MB     04 - Canción para la Magdalena.mp3
Sat Jan 31 17:25:03 2009     5MB     05 - Dieguitos y Mafaldas.mp3
Sat Jan 31 17:25:23 2009     6MB     06 - A mis Cuarenta y Diez.mp3
Sat Jan 31 17:25:35 2009    4MB     07 - El Caso de la Rubia Platino.mp3
Sat Jan 31 17:25:44 2009     3MB     08 - Donde habita el Olvido.mp3
Sat Jan 31 17:25:58 2009     4MB     09 - Cerrado por Derribo.mp3
Sat Jan 31 17:26:19 2009     6MB     10 - Pero que hermosas eran.mp3
Sat Jan 31 17:26:35 2009     4MB     11 - De Purisima y Oro.mp3
Sat Jan 31 17:27:02 2009     7MB     12 - Como te digo una co.mp3
Sat Jan 31 17:27:17 2009     4MB     13 - Noches de Boda.mp3

Fri Jan 23 17:29:47 2009     -> videos/
Fri Jan 23 17:29:36 2009     692MB     American Gangster.avi
Fri Jan 23 17:29:37 2009     692MB     American History X.avi
Fri Jan 23 17:29:37 2009     683MB     Babel.avi
Fri Jan 23 17:29:38 2009     629MB     Cadena Perpetua.avi
Fri Jan 23 17:29:38 2009     673MB     Ciudad de Dios.avi
Fri Jan 23 17:29:39 2009     723MB     El Pianista.avi
Fri Jan 23 17:29:40 2009     654MB     El Ultimo Mohicano.avi
Fri Jan 23 17:29:41 2009     674MB     Enemigo a las Puertas.avi
```

Fri Jan 23 17:29:41 2009	732MB	La Gran Evasion.avi
Fri Jan 23 17:29:41 2009	592MB	La muerte tenia un precio.avi
Fri Jan 23 17:29:42 2009	582MB	La vida es bella.avi
Fri Jan 23 17:29:42 2009	793MB	Los intocables de Eliot Ness.avi
Fri Jan 23 17:29:43 2009	663MB	Memento.avi
Fri Jan 23 17:29:44 2009	574MB	Mystic River.avi
Fri Jan 23 17:29:45 2009	764MB	Pulp Fiction.avi
Fri Jan 23 17:29:45 2009	642MB	Seven.avi
Fri Jan 23 17:29:46 2009	762MB	Sin perdon.avi
Fri Jan 23 17:29:46 2009	562MB	Snatch.avi
Fri Jan 23 17:29:47 2009	762MB	Wall-e.avi

12.2. Instalación y configuración

El proyecto se presenta al usuario como dos paquetes debian. Uno corresponde al cliente y otro al servidor. El nombre de estos paquetes es c3-0.1_i386.deb y c3_Server-0.1_i386.deb respectivamente.

Para la instalación de dichos paquetes se ejecutarán los siguientes comandos:

sudo dpkg -i c3_Server-0.1_i386.deb
sudo dpkg -i c3-0.1_i386.deb
según sea para el servidor o para el cliente.

Lo que hacen estos comandos es instalar ambos programas en la ruta /usr/share/c3-server y /usr/share/c3-client. Ademas crean enlaces simbólicos en /usr/bin que permitirán ejecutar el programa desde cualquier directorio mediante los comandos “c3” para el cliente y “c3-server”.

Además, los dos archivos de configuración, tanto el del cliente como el del servidor, se copiarán en el directorio /etc. Para cualquier modificación en la configuración se deberán editar manualmente dichos ficheros.

12.3. Recomendaciones de uso

Para el total aprovechamiento de las capacidades del sistema, se recomienda renombrar los objetos con nombres significativos para asi poder recuperarlos de manera eficiente.

La idea es sustituir las típicas carpetas de los distintos sistemas operativos por nombres que clasifiquen el objeto. Por ejemplo si queremos subir una canción del disco 19 dias y 500 noches de Joaquin Sabina, un nombre ideal sería algo parecido a lo siguiente:

Joaquin Sabina.19 dias y 500 noches.Cerrado por derribo.mp3

Con este nombre podremos filtrar a la hora de buscar objetos. Por ejemplo si queremos ver qué discos de Sabina tenemos en el bucket "musica" haremos lo siguiente:

c3 lb -b musica -px "Joaquin Sabina"

y esto nos devolvera todas las canciones de todos los discos.

Si solo queremos las canciones del disco 19 dias y 500 noches solo tendremos que precisar el prefijo:

c3 lb -b musica -px "Joaquin Sabina.19 dias y 500 noches"