1º Cuatrimestre 2013 Ejercicio Nº 5 - Archivos Ubicuos	Taller de Programación I (75.42) Facultad de Ingeniería Universidad de Buenos Aires

Trabajo Práctico Final - Archivos Ubicuos

Introducción

Se propone realizar un servicio similar al del conocido producto DropBox[1], que se denominará ArchivosUbicuos (AU). Este servicio debe permitir que los distintos usuarios tengan, cada uno en su computadora, un directorio (que a los fines de la aplicación se denominará “directorio de AU”) cuyo contenido debe sincronizarse automáticamente con el directorio de AU del usuario en otras computadoras.

Descripción

El servicio se compondrá de las siguientes aplicaciones:

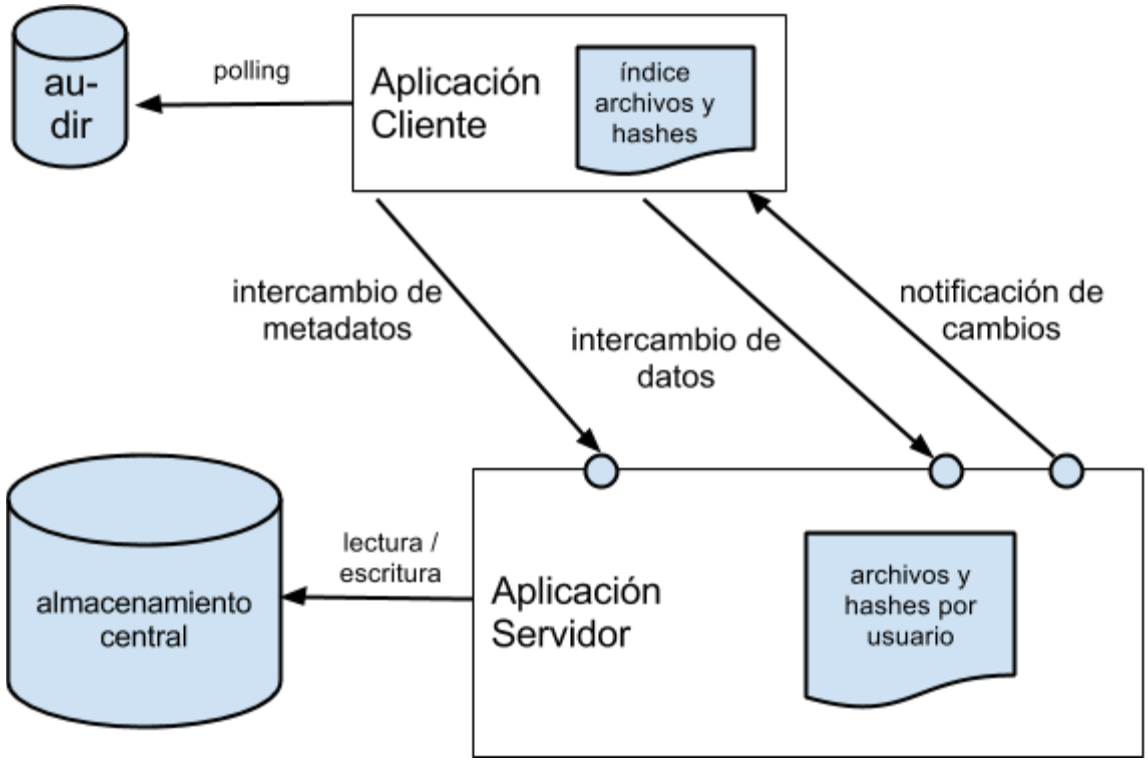
Aplicación cliente: corre en las computadoras de los usuarios. Debe contar con la posibilidad de configurar el acceso al servicio de AU (usuario y contraseña) así como también el directorio que se sincronizará. Dado que para este desarrollo no se contará con un host fijo de contacto para el servidor, se permitirá además configurar el IP del servidor al que el cliente debe conectarse. De acuerdo a las decisiones de arquitectura que realice el grupo, también se deberán configurar números de puertos, intervalo de *polling*, etc. La aplicación deberá contar con una pequeña interfaz gráfica para efectuar estas configuraciones, y deberá soportar su persistencia en disco.

Aplicación servidor: corre en una computadora determinada. Se encarga de mantener el registro de usuarios, de computadoras de dichos usuarios con sus estados de conexión, y de archivos. Además, almacena una copia de los mismos, para que cuando se conecta un cliente pueda sincronizarse. Esta aplicación debe permitir la realización de configuraciones (tales como puertos, ruta donde almacenar archivos) en un archivo del tipo *properties* que se ubicará en un lugar previamente conocido.

Aplicación monitor: corre en la misma computadora que la aplicación servidor y permite realizar las mismas configuraciones que se encuentran en el archivo de *properties* del servidor en forma gráfica, y generar un gráfico de cantidad de bytes almacenados en función del tiempo (actualizado cada un período a determinar por el grupo). Además, contiene la funcionalidad necesaria para gestionar altas, bajas y modificaciones de los usuarios que acceden al servicio.

Esquema del servicio

A continuación se ilustra una idea de cómo podría implementarse el servicio, a muy alto nivel y basado en [2]. Por supuesto, esto es sólo un acercamiento orientativo y el grupo puede (y debe) proponer su solución particular, que puede diferir de lo aquí mostrado tanto como el grupo lo estime conveniente.



En la imagen, se ilustra en primer lugar el *polling* desde la aplicación cliente a su almacenamiento local. El *polling* implica listar los archivos del directorio, recalcular el hash para cada uno, y actualizar el índice local (se aceptan propuestas de optimización basadas en tamaño de archivo, fecha de modificación, etc.).

En segundo lugar, se ilustra la comunicación entre cliente y servidor. Esta comunicación se realiza a un puerto bien definido en el servidor. Un tipo de comunicación que se ha denominado “intercambio de metadatos” (por ejemplo, pedidos de información sobre archivos); otro tipo de comunicación es el denominado intercambio de datos (*upload* y download de *bytes*). Un evento de relevancia, en cuanto a metadatos, para la imagen mostrada es la acción de registro (o desregistro) de la conexión de un cliente, para suscribirse a notificaciones (por ejemplo el agregado o cambio de un archivo). Esto permite que el cliente no tenga que hacer un *polling* constante al servidor. Se requiere que se implemente un mecanismo de notificaciones para hacer un mejor uso de la red al evitar la consulta constante por parte de las aplicaciones cliente.

Las decisiones sobre seguridad involucran varias dimensiones. Por un lado, las aplicaciones (tanto cliente como servidor) guardan información de claves en disco. Por otro lado, la comunicación entre clientes y servidor puede o no ser inteligible para otros que intervengan en el canal, y puede o no estar autenticada (emisor con identidad verificada) y tener un chequeo de integridad (que permita saber que el mensaje no ha sido alterado). Todos estos aspectos se pueden considerar mediante distintas técnicas, permitiendo obtener distintos niveles de seguridad. Para este trabajo, se requiere al menos la verificación de autenticidad del emisor e integridad del mensaje (es decir, que si el mensaje es interceptado podrá ser leído, pero no alterado), para lo que se recomienda tener en cuenta como técnica posible HMAC [3] usando como clave la contraseña del usuario (que es conocida por ambas partes) para generar una firma que sea suministrada con el mensaje. Se sugiere implementar la aplicación de forma incremental, en este sentido la segurización del canal no sería lo primero a implementar durante el desarrollo. También se espera que los archivos que se guarden en disco se configuren con los permisos mínimos necesarios (por ejemplo, si la aplicación cliente almancena credenciales del usuario en un archivo, este archivo no debería ser legible para otros usuarios).

En esencia, el desarrollo el servicio implica el diseño de un protocolo propietario de comunicación. Asumiendo que tal protocolo existe, el siguiente esquema ilustra el funcionamiento conforme al diagrama mostrado:

(el cliente1 inicia su ejecución y actualiza su índice)
cliente1> REGISTER user=johndoe, pass=clave
server> OK
cliente1> GET FILE LIST
server> archivo1.txt #hash; archivo2.txt #hash; archivo3.txt #hash
(el cliente1 compara la lista de archivos y hashes con su índice y determina que archivo1.txt es el mismo que tiene localmente, archivo2.txt ha sido modificado y archivo3.txt es nuevo)
cliente1> GET archivo2.txt
server> data archivo2.txt
cliente1> GET archivo3.txt
server> data archivo3.txt
(el cliente2 inicia su ejecución y actualiza su índice)
cliente2> REGISTER user=johndoe, pass=clave
server> OK
cliente2> GET FILE LIST
server> archivo1.txt #hash; archivo2.txt #hash; archivo3.txt #hash
(el cliente2 compara la lista de archivos y hashes con su índice y determina que no hay cambios respecto de su copia local)
cliente3> REGISTER user=johndoe, pass=clave
cliente3> GET FILE LIST
server> archivo1.txt #hash; archivo2.txt #hash; archivo3.txt #hash
(el cliente3 compara la lista de archivos y hashes con su índice y determina que no hay cambios respecto de su copia local)

(el cliente3 detecta en su ciclo de polling que se modificó archivo1.txt, en este caso se ha elegido enviar también el hash, para que se calcule una sola vez y debido a que tanto cliente como servidor lo necesitan, pero esta decisión queda a cargo del grupo)
cliente3> PUT archivo1.txt #hash - data
server> OK
server> NOTIFY cliente1 - archivo1.txt
server> NOTIFY cliente2 - archivo1.txt
cliente1> GET archivo1.txt
server> data archivo1.txt
cliente2> GET archivo1.txt
server> data archivo1.txt
cliente1> UNREGISTER
cliente2> UNREGISTER
cliente3> UNREGISTER

(en caso que los clientes no se desregistren, igualmente se los debe depurar cada cierto tiempo)

Restricciones y sugerencias

Cada usuario podrá tener un solo directorio de AU en una computadora determinada. La ruta a este directorio, así como las credenciales de autenticación en el servicio, se almacenarán según criterio del grupo. Por ejemplo, se podrían almacenar estos datos en un archivo o directorio oculto en el home del usuario (/home/johndoe/.archivosubicuos).

Las aplicaciones deben tener un mecanismo de logging (log4cpp, syslog, o similar) que permita conocer los eventos más relevantes que han sucedido para una eventual sesión de debugging o una auditoría sobre las acciones realizadas.

Puede ser interesante para el grupo conocer la API REST que ofrece DropBox [4].

La detección de cambios respecto de la copia almacenada en el servidor es una tarea compleja. El grupo podrá realizar propuestas al respecto, aunque como condición inicial se espera que se base en un algoritmo de hashing conocido (tal como MD5 o SHA - DropBox emplea SHA256 como parte de su técnica de almacenamiento), con posibles optimizaciones en base a nombre, fecha, tamaño, etc. Una solución posible es entonces basarse en un *hash* del archivo y asumir que se trata de la misma copia si el *hash* coincide (para esta aplicación el riesgo, mínimo, de que dos archivos distintos coincidan en su *hash* puede ser desestimado). Por tanto, la aplicación servidor tendría para cada archivo su valor de hash computado. Asimismo, la aplicación cliente tendría los valores de hash de sus propios archivos, que debería actualizar (para el caso en que se reemplace el archivo por otro del mismo nombre) cada un cierto tiempo.

Para esta versión del servicio no se requieren funcionalidades de carpetas compartidas, encriptación de los archivos propiamente dichos, transferencia *peer-to-peer* entre clientes, ni posibilidad de ver historial de archivos. Tampoco se requiere el particionamiento de archivos en bloques, aunque si un cliente se desconecta en medio de la transferencia de un archivo, el resultado debe ser consistente. En caso de modificarse un archivo, tampoco se pretende que la actualización se haga con el diferencial, pudiendo enviarse el archivo completo nuevamente.

Características opcionales

Sólo se requiere sincronizar el usuario actualmente logueado en la computadora (y se puede asumir que será solo uno). No se requiere almacenar subdirectorios en forma recursiva (puede asumirse que todos los archivos se encontrarán a nivel del directorio de au), aunque es un agregado opcional que puede implementarse (podría utilizarse el path completo como nombre del archivo en las comunicaciones e índices).

El grupo deberá proponer el modo de ejecución de la aplicación cliente. En caso de decidir opcionalmente la generación de un daemon (para por ejemplo iniciar automáticamente el cliente junto con el sistema), debe darse la opción alternativa de ejecución manual a fines de facilitar que los ayudantes correctores eviten alterar las configuraciones de sus computadoras de prueba.

Referencias

[1] https://www.dropbox.com/

[2] http://www.youtube.com/watch?feature=player_embedded&v=PE4gwstWhmc
Charla brindada por Kevin Modzelewski, de DropBox, en un curso de Stanford University

[3] http://en.wikipedia.org/wiki/Hash-based_message_authentication_code

[4] https://www.dropbox.com/developers/core/api