

En este documento se presenta una breve descripción del funcionamiento de dos importantes sistemas de ficheros distribuidos, **NFS** y **Coda**. Se pretende que através del documento el lector adquiera los conocimientos necesarios para poder comparar ambos sistemas, aunque en el documento no se presenta una comparación explícita de ambos ya que las diferencias son obvias.

## Índice General

<b>1</b>	<b>Sistemas de ficheros distribuidos</b>	<b>2</b>
<b>2</b>	<b>El sistema de archivos de red (NFS) de Sun.</b>	<b>3</b>
2.1	Arquitectura de NFS. . . . .	3
2.2	Protocolos de NFS. . . . .	4
2.3	Implantación de NFS . . . . .	7
<b>3</b>	<b>¿ Qué es Coda ?</b>	<b>9</b>
3.1	¿ Porqué es tan prometedor y potencialmente tan importante? . . . . .	9
3.2	Actividades actuales en Coda. . . . .	10
<b>4</b>	<b>Introducción a Coda</b>	<b>10</b>
4.1	Familiarizándose con la terminología de Coda. . . . .	11
4.2	Organización del cliente. . . . .	13
4.2.1	Los módulos del kernel y el administrador del caché. . . . .	13
4.3	Organización del servidor. . . . .	14
4.3.1	Utilidades. . . . .	14
<b>5</b>	<b>Coda.</b>	<b>14</b>
5.1	Coda en un cliente. . . . .	14

5.2	De la caché al funcionamiento desconectado. . . . .	17
5.3	Volúmenes, Servidores y Replicación de Servidores. . . . .	19
5.4	Coda en acción. . . . .	21
<b>6</b>	<b>Bibliografía.</b>	<b>22</b>

## Índice de Figuras

1	Cliente/Venus/Vice. . . . .	16
2	Clientes y servidores en distintos estados. . . . .	18

## 1 Sistemas de ficheros distribuidos

Un sistema de ficheros distribuido almacena archivos en uno o más ordenadores denominados servidores y los hace accesibles a otros ordenadores denominados clientes, donde se manipulan como si fueran locales. Existen muchas ventajas en el uso de servidores de ficheros : Los archivos están más accesibles si desde varios ordenadores se puede acceder a los servidores, y compartir los archivos de una única localización es más sencillo que distribuir copias de los archivos a todos los clientes. Las copias de respaldo y la seguridad son más fáciles de manejar cuando sólo hay que tener en cuenta a los servidores. Los servidores pueden ofrecer un gran espacio de almacenamiento que sería costoso y poco práctico suministrar a cada cliente. La utilidad de un sistema de ficheros distribuido se vé claramente cuando se considera a un grupo de empleados que tienen que compartir documentos. Poe ejemplo, para compartir programas también es un buen candidato. En ambos casos la administración del sistema se simplifica.

Hay muchos problemas a los que nos enfrentamos en el diseño de un buen sistema de ficheros distribuido. Transportar muchos archivos sobre la red puede facilmente acabar en bajas prestaciones y alta latencia, cuellos de botella en la red y sobrecarga también pueden aparecer. La seguridad de los datos también es un aspecto importante: cómo se puede estar seguro de que un cliente está realmente autorizado para acceder a la información y cómo se puede evitar que los datos sean espiados a través de la red. Otro dos problemas importantes a la hora de atacar el diseño están relacionados con los fallos. A menudo, los ordenadores cliente son más fiables que la red que los conecta y los fallos de la red pueden provocar la inoperatividad del cliente. De manera similar, una fallo del servidor puede ser crítico, ya que puede dejar a todos los clientes sin acceso a información crucial. El proyecto **Coda** ha puesto

especial atención en muchos de estos puntos y los ha implementado como un prototipo de investigación.

## 2 El sistema de archivos de red (NFS) de Sun.

En esta sección se analiza un ejemplo de sistema de archivo de red, el sistema de archivo de red de **Sun Microsystems**, conocido universalmente como **NFS** que fue diseñado e implantado por **Sun** en un principio para su uso en las estaciones de trabajo *UNIX*. Ahora, otros fabricantes lo soportan, tanto para *UNIX* como para otros sistemas operativos (incluido *MS-DOS*). **NFS** soporta sistemas heterogéneos; por ejemplo, clientes de *MS-DOS* que utilizan servidores *UNIX*. Ni siquiera es necesario que todas las máquinas utilicen el mismo hardware. Es común encontrar clientes de *MS-DOS* que utilizan CPU con **Intel 386** y obtienen algún servicio de los servidores de archivos de *UNIX* que se ejecutan en **Motorola 68030** o **CPU SPARC de Sun**.

Tres aspectos de NFS son de interés: la arquitectura, el protocolo y la implantación. Se analizan a continuación:

### 2.1 Arquitectura de NFS.

La idea básica detrás de **NFS** es permitir que una colección arbitraria de clientes y servidores compartan un sistema de archivos común. En la mayoría de los casos, todos los clientes y servidores están en la misma **LAN**, pero esto no es necesario. Es posible ejecutar **NFS** en una red de área amplia. Para simplificar la exposición se hablará de los clientes y servidores como si estuvieran en máquinas diferentes, pero de hecho, **NFS** permite que cada máquina sea cliente y servidor al mismo tiempo.

Cada servidor **NFS** exporta uno o más de sus directorios para su acceso por parte de los clientes remotos. Cuando se dispone de un directorio, también de todos sus subdirectorios, así que, de hecho, los árboles de directorios se exportan como una unidad. La lista de directorios que puede exportar un servidor se conserva en el archivo */etc/exports*, de modo que estos directorios se pueden exportar de manera automática al arrancar el servidor.

Los clientes tienen acceso a los directorios exportados al montarlos. Cuando un cliente monta un directorio (remoto), éste se vuelve parte de su jerarquía de directorios. Muchas estaciones de trabajo **Sun** no tienen disco. Si así lo desea, un cliente sin disco puede montar un sistema de archivos remoto en su directorio raíz, lo que produce un sistema de archivos por completo soportado en un servidor remoto. Las estaciones de trabajo que sí tienen discos locales pueden montar directorios remotos en el sitio que deseen de su jerarquía local de directorios,

lo que produce un sistema de archivos parcialmente local y parcialmente remoto. Para que los programas se ejecuten en la máquina cliente (casi) no existe diferencia entre un archivo localizado en un servidor de archivos remoto y un archivo localizado en el disco local.

Así, la característica básica de la arquitectura de **NFS** es que los servidores exportan directorios y los clientes los montan de manera remota. Si dos o más clientes montan el mismo directorio al mismo tiempo, se pueden comunicar compartiendo archivos en sus directorios comunes. Un programa en un cliente puede crear un archivo, y un programa en otro cliente puede leer el archivo. Una vez realizados los montajes, no hay que hacer nada especial para lograr compartir los archivos. Los archivos compartidos sólo están ahí, en la jerarquía de directorios de varias máquinas y pueden leerse y escribir en ellos de la manera usual. Esta sencillez es uno de los puntos fuertes de **NFS**.

## 2.2 Protocolos de NFS.

Puesto que uno de los objetivos de **NFS** es soportar un sistema heterogéneo, con clientes y servidores que tal vez ejecuten diferentes sistemas operativos con un hardware distinto, es esencial que la interfaz entre los clientes y los servidores esté bien definida. Sólo entonces es posible que cualquiera pueda escribir una nueva implantación cliente y espere que funcione de manera correcta con los servidores existentes y viceversa.

**NFS** logra este objetivo al definir dos protocolos **cliente-servidor**. Un protocolo es un conjunto de solicitudes enviadas por los clientes a los servidores, junto con las respuestas enviadas de regreso de los servidores a los clientes. Mientras un servidor reconozca y pueda controlar todas las solicitudes en los protocolos, no necesita saber nada de sus clientes. De manera análoga, los clientes pueden tratar a los servidores como "*cajas negras*" que aceptan y procesan un conjunto específico de solicitudes. La forma en que lo hacen es asunto de ellos.

El primer protocolo **NFS** controla el anclaje. Un cliente puede enviar un nombre de ruta de acceso a un servidor y solicitar que monte ese directorio en alguna parte de su jerarquía de directorios. El lugar donde se montará no está contenido en el mensaje, ya que el servidor no se preocupa por dicho lugar. Si la ruta es válida y el directorio especificado ha sido exportado, el servidor regresa un **identificador de archivo** al cliente. El asa de archivo contiene campos que identifican de manera única al tipo de sistema de archivo, el disco, el número de **inodo** del directorio, e información de seguridad. Las llamadas posteriores para la lectura y escritura de archivos en el directorio montado utilizan el identificador del archivo.

Muchos clientes están configurados de modo que monten ciertos directorios remotos sin intervención manual. Por lo general, estos clientes contienen un archivo llamado */etc/rc*, que es un *shellscript* que contiene las instrucciones para el montaje remoto. Este *shellscript*

se ejecuta de manera automática cuando el cliente se arranca.

Otra alternativa a la versión de UNIX de Sun soporta también el **automontaje**. Esta característica permite asociar un conjunto de directorios con un directorio local. Ninguno de los directorios remotos se monta (ni se realiza el contacto con sus servidores) cuando arranca el cliente. En vez de esto, la primera vez que se abre un archivo remoto, el sistema operativo envía un mensaje a cada uno de los servidores. El primero en responder gana, y se monta su directorio.

El automontaje tiene dos ventajas principales sobre el montaje estático por medio del archivo */etc/rc*. La primera es que si uno de los servidores **NFS** llamados en */etc/rc* no está sirviendo peticiones, es imposible despertar al cliente, al menos no sin cierta dificultad, retraso y unos cuantos mensajes de error. Si el usuario ni siquiera necesita ese servidor por el momento, todo ese trabajo se desperdicia. En segundo lugar, al permitir que el cliente intente comunicarse con un conjunto de servidores en paralelo, se puede lograr cierto grado de tolerancia a fallos (puesto que sólo se necesita que uno de ellos esté activo) y se puede mejorar el desempeño (al elegir el primero que responda, que supuestamente tiene la menor carga).

Por otro lado, se supone de manera implícita que todos los sistemas de archivos especificados como alternativas para el automontaje son idénticos. Puesto que **NFS** no da soporte para la réplica de archivos o directorios, el usuario debe lograr que todos los sistemas de archivos sean iguales. En consecuencia, el automontaje se utiliza con más frecuencia para los sistemas de archivos exclusivos para lectura que contienen binarios del sistema y para otros archivos que rara vez cambian.

El segundo protocolo **NFS** es para el acceso a directorios y archivos. Los clientes pueden enviar mensajes a los servidores para que manejen los directorios y lean o escriban en archivos. Además, también pueden tener acceso a los atributos de un archivo, como el modo, tamaño y tiempo de su última modificación. La mayor parte de las llamadas al sistema *UNIX* son soportadas por **NFS**, con la probable sorpresa de **OPEN** y **CLOSE**.

La omisión de **OPEN** y **CLOSE** no es un accidente. Es por completo intencional. No es necesario abrir un archivo antes de leerlo, o cerrarlo al terminar. En vez de esto, para leer un archivo, un cliente envía al servidor un mensaje con el nombre del archivo, una solicitud para buscarlo y regresar un **identificador de archivo**, que es una estructura de identificación del archivo. A diferencia de una llamada **OPEN**, esta operación **LOOKLJP** no copia información a las tablas internas del sistema. La llamada **READ** contiene al identificador de archivo que se desea leer, el ajuste para determinar el punto de inicio de la lectura y el número de bytes deseados. Cada uno de estos mensajes está autocontenido. La ventaja de este esquema es que el servidor no tiene que recordar lo relativo a las conexiones abiertas entre las llamadas a él. Así, si un servidor falla y después se recupera, no se pierde información acerca de los archivos abiertos, puesto que no hay ninguno. Un servidor como

éste, que no conserva información del estado de los archivos abiertos se denomina **sin estado**.

Por el contrario, en el **sistema V** de *UNIX*, el sistema de archivos remotos (**RFS**) requiere abrir un archivo antes de leerlo o escribir en él. El servidor crea entonces una entrada de tabla con un registro del hecho de que el archivo está abierto y la posición actual del lector, de modo que cada solicitud no necesita un ajuste. La desventaja de este esquema es que si un servidor falla y vuelve a arrancar rápidamente, se pierden todas las conexiones abiertas, y fallan los programas cliente. **NFS** no tiene esta característica.

Por desgracia, el método **NFS** dificulta el hecho de lograr la semántica de archivo propia de *UNIX*. Por ejemplo, en *UNIX* un archivo se puede abrir y bloquear para que otros procesos no tengan acceso a él. Al cerrar el archivo, se liberan los bloqueos. En un servidor sin estado como **NFS**, las cerraduras no se pueden asociar con los archivos abiertos, puesto que el servidor no sabe cuáles son los archivos están abiertos. Por lo tanto, **NFS** necesita un mecanismo independiente adicional para controlar los bloqueos.

**NFS** utiliza el mecanismo de protección de *UNIX*, con los bits **rwX** para el propietario, grupo y demás personas. En un principio, cada mensaje de solicitud sólo contenía los identificadores del usuario y del grupo de quien realizó la llamada, lo que utilizaba el servidor **NFS** para validar el acceso. De hecho, confiaba en que los clientes no mintieran. Varios años de experiencia han demostrado ampliamente que tal hipótesis era un tanto ingenua. Actualmente, se puede utilizar la criptografía de claves públicas para establecer una clave segura y validar al cliente y al servidor en cada solicitud y respuesta. Cuando esta opción se activa, un cliente malicioso no puede personificar a otro cliente, pues no conoce la clave secreta del mismo. Por cierto, la criptografía sólo se utiliza para autenticar a las partes. Los datos nunca se encriptan.

Todas las claves utilizadas para la autenticación, así como la demás información, son mantenidas por el **NIS** (servicio de información de la red). **NIS** se conocía antes como el directorio de *páginas amarillas* (*yellow pages*). Su función es la de guardar parejas (*clave, valor*). Cuando se proporciona una clave, regresa el valor correspondiente. No sólo controla las claves de cifrado, sino también la asociación de los nombres de usuario con las contraseñas (cifradas), así como la asociación de los nombres de las máquinas con las direcciones de la red, y otros elementos.

Los servidores de información de la red se duplican mediante un orden **maestro/esclavo**. Para leer sus datos, un proceso puede utilizar al maestro o cualquiera de sus copias (esclavos). Sin embargo, todas las modificaciones deben ser realizadas únicamente en el maestro, que entonces las propaga a los esclavos. Existe un breve intervalo después de una actualización en el que la base de datos es inconsistente.

## 2.3 Implantación de NFS

Aunque la implantación del código del cliente y el servidor es independiente de los protocolos **NFS**, es interesante echar un vistazo a la implantación de **NFS**. La capa superior es la capa de llamadas al sistema, la cual controla las llamadas como **OPEN**, **READ** y **CLOSE**. Después de analizar la llamada y verificar sus parámetros, llama a la segunda capa, la capa del sistema virtual de archivos (**VFS**).

La tarea de la capa **VFS** es mantener una tabla con una entrada por cada archivo abierto, análoga a la tabla de **inodos** para los archivos abiertos en *UNIX*. En el *UNIX* ordinario, un **inodo** queda indicado de manera única mediante una pareja (*dispositivo, inodo*). En vez de esto, la capa **VFS** tiene una entrada, llamada **vnodo** (**inodo** virtual), para cada archivo abierto. Los **vnodos** se utilizan para indicar si un archivo es local o remoto. Para los archivos remotos, se dispone de suficiente información para tener acceso a ellos.

Para ver la forma de utilizar los **vnodos**, sigamos una secuencia de llamadas al sistema **MOUNT**, **OPEN** y **READ**. Para montar un sistema remoto de archivos, el administrador del sistema llama al programa **mount** con la información del directorio remoto, el directorio local donde será montado y algunos otros datos adicionales. El programa **mount** analiza el nombre del directorio remoto por montar y descubre el nombre de la máquina donde se localiza dicho directorio. Entonces, entra en contacto con la máquina en la que se localiza el directorio remoto. Si el directorio existe y está disponible para su montaje remoto, el servidor regresa entonces un *identificador de archivo* para el directorio. Por último, llama a **MOUNT** para transferir el *identificador del archivo* al núcleo.

El núcleo construye entonces un **vnodo** para el directorio remoto y pide el código del cliente **NFS** para crear un **rnodo** (**inodo** remoto) en sus tablas internas, con el fin de mantener el *identificador del archivo*. El **vnodo** apunta al **rnodo**. Así, cada **vnodo** de la capa **VFS** contendrá en última instancia una referencia a un **rnodo** en el código del cliente **NFS** o una referencia a un **inodo** en el sistema operativo local. Así, es posible ver desde el **vnodo** si un archivo o directorio es local o remoto y, si es remoto, encontrar su *identificador de archivo*.

Al abrir un archivo remoto, en cierto momento durante el análisis del nombre de la ruta de acceso, el núcleo alcanza el directorio donde se desea montar el sistema de archivos remoto. Ve que este directorio es remoto y en el **vnodo** del directorio encuentra la referencia al **rnodo**. Le pide entonces al código del cliente **NFS** que abra el archivo. El código del cliente **NFS** busca en la parte restante del nombre de la ruta de acceso en el servidor remoto asociado con el directorio montado y regresa un identificador de archivo para él. Crea en sus tablas un **rnodo** para el archivo remoto y regresa a la capa **VFS**, la cual coloca en sus tablas un **vnodo** para el archivo que apunta al **rnodo**. De nuevo, vemos aquí que todo archivo o directorio abierto tiene un **vnodo** que apunta a un **rnodo** o a un **inodo**.

Quien hizo la llamada recibe un descriptor de archivo para el archivo remoto. Este descriptor de archivo se asocia con el **vnodo** mediante las tablas en la capa **VFS**. Observe que no se crean entradas en las tablas del lado del servidor. Aunque el servidor está listo para proporcionar los *identificadores de archivo* que le soliciten, no mantiene un registro de los archivos que tienen *identificadores activos* y los que no. Cuando se le envía un asa de archivo para el acceso a un archivo, verifica el *identificador* y, si éste es válida, se utiliza. El proceso de validación puede incluir una clave de autenticación contenida en los encabezados **RPC**, si la seguridad está activada.

Cuando el descriptor de archivo se utiliza en una llamada posterior al sistema, por ejemplo, **READ**, la capa **VFS** localiza el **vnodo** correspondiente y por medio de él determina si es local o remoto y el **inodo** o **rnodo** que lo describe.

Por razones de eficiencia, las transferencias entre el cliente y el servidor se realizan en bloques grandes, por lo general de 8192 bytes, aunque se soliciten menos. Después de que la capa **VFS** del cliente ha obtenido el bloque de 8K que necesitaba, emite en forma automática una solicitud del siguiente bloque, por lo que lo recibirá rápidamente. Esta característica se conoce como **lectura adelantada** y mejora en forma considerable el rendimiento.

Se sigue una política análoga para la escritura. Si una llamada **WRITE** proporciona menos de 8192 bytes de datos, los datos se acumulan en forma local. Sólo cuando el último pedazo de 8K está completo, se envía al servidor. Sin embargo, al cerrar un archivo, todos sus datos se envían al servidor de manera inmediata.

Otra de las técnicas que se utilizan para mejorar el rendimiento es el ocultamiento, como en el *UNIX* ordinario. Los servidores ocultan los datos para evitar el acceso al disco, pero esto es invisible para los clientes. Los clientes mantienen dos caches, uno para los atributos de archivo (**inodos**) y otro para los datos del archivo. Cuando se necesita un **inodo** o un bloque del archivo, primero hay que verificar si esta solicitud se puede satisfacer mediante el caché del cliente. En este caso, se evita el tráfico en la red.

Aunque el ocultamiento por parte del cliente ayuda en mucho a la tarea, también presenta algunos problemas molestos. Supongamos que dos clientes ocultan el mismo bloque del archivo y que uno de ellos lo modifica. Cuando el otro lee el bloque, obtiene el valor antiguo. El caché no es coherente.

Debido a la severidad potencial de este problema, la implantación de **NFS** hace varias cosas para mitigarlo. Una de ellas es que a cada bloque caché se le asocia un cronómetro. Cuando éste expira, la entrada se descarta. Por lo general, el tiempo es de 3 segundos para los bloques de datos y de 30 segundos para los bloques de directorio. Esto reduce un poco el riesgo. Además, al abrir un archivo con caché, se envía un mensaje al servidor para revisar la hora de la última modificación. Si la última modificación ocurrió antes de capturar en el

caché la copia local, se descarta la copia del caché y se utiliza la nueva copia del servidor. Por último, el cronómetro del caché expira cada 30 segundos y todos los bloques sucios (es decir, modificados) en el caché se envían al servidor.

Aún así, **NFS** ha recibido amplias críticas por no implantar de manera adecuada la semántica apropiada de **UNIX**. Una escritura a un archivo de un cliente podría o no ser vista cuando otro cliente lea el archivo, según la sincronización. Además, al crear un archivo, esta acción podría no ser visible para el mundo exterior durante un periodo de 30 segundos. Existen otros problemas similares.

Por medio de este ejemplo, vemos que aunque **NFS** tiene un sistema compartido de archivos, como el sistema resultante es una especie de **UNIX** parchado, la semántica del acceso a los archivos no está por completo bien definida y la ejecución de un conjunto de programas que cooperen entre sí podría producir diversos resultados, según la sincronización. Además, lo único con lo que trata **NFS** es con el sistema de archivos. No hace referencia a otros aspectos, como la ejecución de un proceso. A pesar de todo, **NFS** es popular y tiene un uso muy extendido.

### 3 ¿ Qué es Coda ?

Un sistema de ficheros en red avanzado cuyo origen está en el sistema de ficheros **AFS2**. Ha sido diseñado en la **CMU** (Carnegie Mellow University), durante los últimos años, por el grupo del Sr. Satyanarayanan en el departamento **SCS** ( School of Computer Sciencie).

#### 3.1 ¿ Porqué es tan prometedor y potencialmente tan importante?

Porque tiene varias características muy destacables en los sistemas de ficheros en red. Actualmente de estas características no se encuentran en ningún otro sistema.

- Altas prestaciones a través de un caché persistente en el lado del cliente.
- Replicación en varios servidores.
- Buena escalabilidad.
- Adaptación al ancho de banda.
- Funcionamiento continuado durante fallos producidos en la red del servidor.
- Posibilidad de funcionamiento desconectado para procesadores móviles.

- Modelo de seguridad para autenticación, encriptación y control de acceso.
- Semántica bien definida de compartición, incluso con fallos en la red.
- Disponible de manera gratuita.

### 3.2 Actividades actuales en Coda.

El CMU está realizando un importante esfuerzo para mejorar **Coda**. Es importante llevar el sistema de su estado actual a un gran número de sistemas y hacer especial hincapié en:

- Mejorar las prestaciones y la fiabilidad.
- Portarlo a distintas plataformas.
- Documentación y grupos de noticias.
- Extensiones en su funcionalidad.

**Coda** fue originalmente implementado sobre **Match 2.6** y recientemente ha sido portado a *FreeBSD*, *Linux* y *NetBSD*. Michael Callahan ha portado una gran parte de **Coda** a *Windows'95/98* y se está estudiando *Windows NT*. Actualmente el gran esfuerzo del equipo de **Coda** está en portar el sistema y en hacerlo más robusto. Unas pocas características nuevas están siendo implementadas (caché de tipo **write-back** y acceso a múltiples celdas por ejemplo) y en muchas areas, los componentes de **Coda** están siendo reorganizados. Se ha recibido una generosa ayuda de muchos de los usuarios de la red y se espera que esto continúe. Quizás **Coda** se convierta en un sistema de ficheros distribuidos popular, ampliamente usado, y disponible de forma gratuita.

## 4 Introducción a Coda

**Coda** es un sistema de ficheros distribuido, esto es, hace que los archivos estén disponibles a un conjunto de ordenadores clientes como si éstos formarían parte de su árbol de directorios, pero manteniendo la autoridad de los datos de los archivos en los servidores. **Coda** contiene algunas características que le hacen destacar: Soporta funcionamiento desconectado, o sea acceso completo a una parte cacheada del espacio de ficheros en el caso de que ocurran desconexiones voluntarias o involuntarias de la red o del servidor. Cuando los clientes vuelvan a estar conectados, **Coda** reintegrará los cambios que se han realizado mientras se trabajaba desconectado. Además **Coda** tiene un sistema de replicación para fallos del servidor, ésto

significa que los datos son almacenados y obtenidos en cualquier servidor dentro de un grupo de servidores y **Coda** continuará funcionando con sólo un subconjunto de esos servidores disponibles. Si aparecen diferencias entre servidores, debidas a particiones de red, **Coda** resolverá estas diferencias automáticamente en la mayor extensión posible y ayudará a los usuarios a reparar aquéllo que no se puede reparar de manera automática. **Coda** está organizado de manera muy diferente a **NFS** o a los directorios compartidos de **Windows/Samba**. **Coda** tiene muchas similitudes con **AFS** y **DCE/DFS**.

## 4.1 Familiarizándose con la terminología de Coda.

### Espacio simple de nombres.

Todo **Coda** aparece bajo un único directorio */coda* (o en un único disco Windows). **Coda** no tiene la posibilidad de realizar distintas exportaciones o comparticiones como hacen **NFS** y **Samba**, que pueden ser montadas individualmente. Bajo */coda* los volúmenes (también llamados conjuntos de ficheros) o archivos exportados por todos los servidores (existentes en la celda coda) son visibles. **Coda** encuentra automáticamente servidores, y cada cliente necesita saber el nombre de un servidor de inicio que le da su información para encontrar el *volumen raíz* de **Coda**.

### Una celda Coda.

Es un grupo de servidores que comparten un conjunto de bases de datos de configuración. Una celda puede estar formada por un único servidor o cientos de ellos. Un servidor se designa como el **SCM**, la máquina que controla el sistema. Se distingue por ser el único servidor que modifica las bases de datos de configuración compartidas por todos los servidores, y propagando esos cambios a los otros servidores. Actualmente un cliente **Coda** sólo puede pertenecer a una celda. Se espera obtener un mecanismo para **Coda** de manera que un cliente pueda acceder a archivos en varias celdas.

### Volúmenes Coda.

Los servidores de archivos agrupan los archivos en *volúmenes*. Un volumen es típicamente más pequeño que una partición, pero más grande que un directorio. Los volúmenes tienen una raíz, y contienen un árbol de directorios con archivos. Cada volumen es montado en algún lugar bajo */coda* y forma un subárbol de */coda*. Los volúmenes pueden tener partes de "montaje" de otros volúmenes. Un punto de montaje de un volumen no es un punto de montaje al estilo UNIX o una unidad de disco de Windows. Existe un único disco o punto de anclaje UNIX para **Coda**. Un punto de anclaje **Coda** contiene suficiente información para que el cliente encuentre los servidores dentro en los cuales almacenan los archivos dentro del volumen. El grupo de servidores que sirve el volumen, se llama el *grupo de almacenamiento del volumen*.

### Puntos de montaje del volumen.

Un volumen especial es el volumen raíz, el volumen que **Coda** monta en */coda*. Otros volúmenes pueden insertarse dentro del árbol **Coda**, en el árbol de directorios de **Coda** usando **cfs mkmount**. Este comando instala un punto de anclaje de un volumen en el árbol de directorios de **Coda**; el resultado sería similar a ejecutar **mkdir directorio; mount dispositivo directorio** en UNIX. Cuando se ejecuta el comando **cfs makemount** es necesario pasar como parámetros dos argumentos que son el nombre del **punto de anclaje** y el **nombre del volumen** a montar. Los puntos de anclaje de **Coda** son objetos persistentes, no como los puntos de anclaje de UNIX que necesitan remontarse después de una reinicialización de la máquina.

### Almacenamiento de datos.

Los servidores no almacenan y exportan volúmenes como un directorio en el sistema de ficheros local como **NFS** y **Samba**. **Coda** necesita muchos más metadatos para soportar la replicación y el funcionamiento desconectado, y tiene un sistema de recuperación complejo que sería difícil de llevar a cabo dentro de un sistema de ficheros local. Los servidores **Coda** almacenan archivos identificados por un número típicamente todos bajo el directorio */vicepa*. Los metadatos (dueños, listas de control de acceso, vectores de versión) y directorios de contenidos se almacenan en un archivo de datos **RVM** que a menudo podría ser una partición de disco sin sistema de archivos (*cruda*).

### RVM

Son las siglas de Recoverable Virtual Memory (Memoria Virtual Recuperable). **RVM** es una biblioteca transaccional que hace que parte de un espacio virtual de direcciones de un proceso persistente en disco almacenen sus cambios a esta memoria de forma atómica para realizar un almacenamiento persistente. **Coda** utiliza **RVM** para manipular sus metadatos. Estos datos se almacenan en un archivo de datos de **RVM** que está mapeado en la memoria desde el arranque. Las modificaciones se realizan en memoria virtual y también son escritos en el archivo de log de la **RVM** cuando se realiza el *commit* de la transacción. El archivo de log contiene los datos a los que se ha hecho un *commit* y que no se han incorporado aún al archivo de datos en el disco.

### Datos del Cliente

Es almacenado de un modo bastante parecido. los metadatos en la **RVM** (normalmente en */usr/coda/DATA*) y los archivos en la caché son almacenados por un número en */usr/coda/venus.cache*. El caché en un cliente es persistente. Este caché contiene copias de los archivos del servidor. El caché permite un acceso rápido para el cliente a los datos y acceder a estos aunque el cliente esté desconectado.

## Validación

Cuando **Coda** detecta que un servidor está disponible de nuevo, validará los datos del caché antes de usarlos para asegurarse de que los datos de la caché pertenecen a la última versión del fichero. **Coda** compara las marcas de las versiones del caché asociadas a cada objeto, con las marcas de las versiones mantenidas en el servidor.

## Autenticación

**Coda** maneja autenticación y autorización a través de un *testigo*. De manera similar (los detalles son muy diferentes) a usar un recurso compartido de windows, **Coda** requiere que los usuarios hagan *login*. Durante el proceso de entrada, el cliente obtiene una clave de entrada (o *testigo*) a cambio de una palabra de paso correcta. El testigo se asocia con una identificación de usuario, actualmente esta identificación de **Coda** es el **UID** del usuario que ha realizado el *login*.

## Protección

Para conceder permisos el administrador de la caché y los servidores usan el *testigo* con su identidad asociada y comparan esto con los privilegios concedidos a esa identidad en la *lista de control de accesos (ACL)*. Si no hay un *testigo* asignado, se asume que el acceso es anónimo, para los cuales, los permisos son otra vez concedidos a través de la *lista de control de accesos* usando la identidad **System:AnyUser**.

## 4.2 Organización del cliente.

### 4.2.1 Los módulos del kernel y el administrador del caché.

Como cualquier otro sistema de ficheros, un ordenador habilitado para usar el sistema de ficheros de **Coda** necesita algún tipo de soporte en el kernel para acceder a los archivos de **Coda**. El soporte para **Coda** del núcleo es mínimo, y trabaja en unión con el administrador de la caché (en el espacio de memoria de usuario) **Venus**. Los programas usuarios realizarán peticiones al núcleo, que responderá directamente o preguntará al administrador de la caché (**Venus**) para que le ayude a ofrecer el servicio.

Normalmente el código para el kernel está en un módulo del núcleo. que puede ser cargado en el momento de la inicialización del sistema o cargado dinámicamente cuando el programa **Venus** se arranca. *Venus* incluso montará el sistema de ficheros de **Coda** en */coda*.

**Utilidades.** Para manipular las listas de control de acceso, la caché, los puntos de anclaje de los volúmenes y posiblemente la funcionalidad de red de un cliente en **Coda** se proveen varias utilidades pequeñas. La más importante es el comando **cfs**.

También hay un programa, **clog**, para identificarse ante el servidor de identificación de **Coda**. El programa **codacon** le permite monitorizar las operaciones del administrador de la caché y **cmon** da una lista de los servidores.

### 4.3 Organización del servidor.

El principal programa del servidor de ficheros **Coda** es **codasrv**. Es responsable de realizar todas las operaciones, así como del servicio de localización de volúmenes.

El servidor de autenticación **auth2** maneja las peticiones de **clog** de los *testigos*, y cambia las palabras de paso de **au** y **cpasswd**. Sólo el proceso **auth2** en el **SCM** modificará la base de datos de palabras de paso.

Todos los servidores de una *celda* **Coda** comparten las bases de datos de configuración de */vice/db* y las recupera del **SCM** cuando se han realizado cambios. El programa **updateclnt** es responsable de recuperar esos cambios y pregunta al **updateserv** en la **SCM** para ver si algo ha cambiado. Algunas veces el **SCM** necesita una base de datos (no compartida) de otro servidor para actualizar una base de datos compartida. La obtiene a través de un proceso **updatesrv** en el servidor que usa **updatefetch**.

#### 4.3.1 Utilidades.

Para el servidor existen utilidades para la creación y manipulación de volúmenes. Esas utilidades consisten en *shellscripts* y el comando **volutil**. Hay también una herramienta para manipular la protección de las bases de datos.

## 5 Coda.

### 5.1 Coda en un cliente.

Si **Coda** se está ejecutando en un cliente, se toma para el ejemplo una estación de trabajo *Linux*, escribiendo **mount** se mostrará qué sistema de ficheros (de tipo **Coda**) está montado bajo */coda*. Todos los archivos que cualquiera de los servidores pueden proveer al cliente, están disponibles bajo este directorio, y todos los clientes ven el mismo espacio de nombres. Un cliente se conecta a **Coda** y no a servidores individuales, que entran en juego de manera transparente. Esto es diferente de la manera en que se monta un sistema de ficheros **NFS**, en cuyo caso se montan sistemas de ficheros por servidor. En un sistema más usual como **Windows** (*Novell* y *CIFS*) también con *Appleshare* en **Macintosh**, también los archivos

se montan por volumen. El espacio global de nombres no es nuevo. El sistema de ficheros de Andrew (*Andrew file system*, **AFS**), predecesor de **Coda**, fué pionero en la idea de almacenar todos los ficheros bajo el directorio */afs*. De manera similar, el sistema de ficheros distribuido **DSF/DCE** de la *OFS*, monta sus ficheros bajo un directorio. El nuevo sistema de archivos distribuidos de Microsoft (**dfs**) provee enlaces para poner todo lo compartido por los servidores bajo un único árbol de directorios, parecido al tipo de enlace provisto por los demonios de auto montaje (*auto-mount*) de unix y el servicio de páginas amarillas. ¿ Por qué se obtienen tantas ventajas de un único punto de montaje ? Porque esto significa que todos los clientes pueden ser configurados de la misma manera, y que los usuarios siempre verán el mismo árbol de directorios. Para grandes instalaciones esto es esencial. Con **NFS** el cliente necesita una lista actualizada de los servidores y de los directorios que estos exportan, mientras que con **Coda** un cliente meramente necesita saber donde encontrar el directorio raíz de **Coda** */coda*. Cuando se añaden nuevos servidores o recursos compartidos, el cliente los descubrirá automáticamente en algún lugar bajo el árbol */coda*.

Para comprender cómo **Coda** puede funcionar cuando las conexiones de red han sido servidas, se analizará una operación simple del sistema de ficheros. Supongamos que se escribe: `"cat /coda/tmp/foo"` para mostrar el contenido de un archivo **Coda**. ¿ Qué ocurre actualmente ? El programa **cat** realiza unas pocas llamadas al sistema con relación al archivo. Una llamada al sistema es una operación a través de la cual un programa pide un servicio del núcleo. Por ejemplo, cuando se abre un archivo, el núcleo realizará una operación de búsqueda para encontrar el **inode** del archivo y devolverá un identificador de archivo asociado al programa que hizo la petición. El **inode** contiene información sobre cómo se accede a los datos del archivo, y esta información es usada por el núcleo, (el identificador del fichero) es necesario para que el programa pueda abrirlo. La llamada **open** hace entrar al núcleo en el sistema virtual de ficheros (**VFS**) y cuando este detecta que la petición es para un fichero en el sistema de ficheros */coda* es manejada por el módulo del sistema de ficheros **Coda** en el núcleo. **Coda** es un módulo de sistema de ficheros con un tamaño mínimo: mantiene una caché de peticiones respondidas recientemente desde el **VFS**, pero por otra parte pasa la petición al administrador de la caché de **Coda**, denominado **Venus**. **Venus** buscará en la caché de disco del cliente a */tmp/foo*, y en caso de que no esté presente contactará con los servidores para pedir a */tmp/foo*. Cuando se localiza el archivo, **Venus** responde al núcleo, que devuelve el control al programa que realizó la llamada al sistema. Todo el proceso se muestra de manera esquemática en la figura 1 (Cliente/Venus/Vice).

La figura muestra cómo un programa usuario pide un servicio del núcleo a través de una llamada al sistema. El kernel la pasa a **Venus**, permitiéndole leer la petición desde el dispositivo de caracteres */dev/cfs0*. **Venus** trata de resolver la petición buscando en su caché, preguntando a los servidores o posiblemente declarando desconexión y sirviéndola en modo desconectado. El modo desconectado entra en funcionamiento cuando no hay conexión

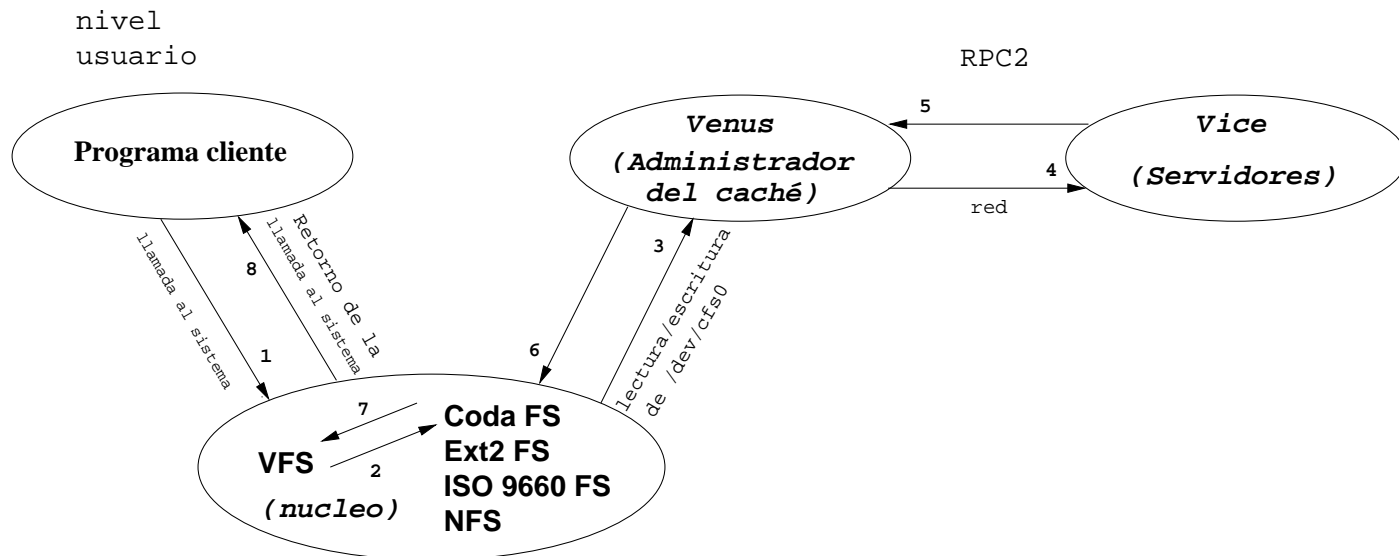


Figura 1: Cliente/Venus/Vice.

de red a cualquier servidor que tenga ficheros. Esto ocurre típicamente cuando los clientes se desconectan de la red o cuando existen fallos de red. Si la operación falla por una desconexión de los servidores también entraría en funcionamiento.

Cuando el núcleo pasa la petición de apertura a **Venus** por primera vez, **Venus** obtiene el fichero entero desde los servidores, usando llamadas a procedimientos remotos para contactar con ellos. Entonces almacena el archivo como un contenedor en la zona de caché (actualmente */usr/coda/venus.cache*). El archivo es ahora un archivo normal en el disco local, y las operaciones de lectura y escritura no llegarían a **Venus** y serían manejadas de manera completa por el sistema de ficheros local (**ext2** en *Linux*). Las operaciones de lectura/escritura de **Coda** funcionarán a la misma velocidad que en los archivos locales. Si el fichero se abre por segunda vez, no se traerá de los servidores otra vez y la copia local estará disponible para su uso inmediatamente. Los archivos de directorio (recordemos que un directorio es únicamente un archivo) así como todos los atributos (dueños, permisos y tamaños) son todos cacheados por **Venus** y **Venus** permite que las operaciones puedan continuar sin contactar con el servidor si los archivos están presentes en la caché. Si el archivo se ha modificado y ha sido cerrado, entonces **Venus** actualiza los servidores enviándoles el nuevo archivo. Otras operaciones que pueden modificar el sistema de archivos, como la creación de directorios, el borrado de archivos o directorios y la creación o borrado (simbólico) de enlaces, también son propagadas al servidor.

Como se puede apreciar, **Coda** cachea toda la información que necesita en el cliente y sólo informa al servidor de los cambios que deben realizarse al sistema de ficheros. Los estudios han demostrado que las modificaciones son bastante menos habituales que el acceso "Sólo

*de lectura*" a los ficheros. Estos mecanismos para cachear datos de una manera *agresiva*" fueron implementados en **AFS** y **DFS**, pero muchos otros sistemas tienen un sistema de caché más rudimentario. Se verá a continuación cómo **Coda** mantiene la consistencia, pero primero indagemos que más se necesita para soportar el funcionamiento desconectado.

## 5.2 De la caché al funcionamiento desconectado.

El origen del funcionamiento desconectado en **Coda** es una de las características originales del proyecto: proveer un sistema de ficheros con capacidad de recuperación ante fallos de red. **AFS**, que soportó miles de clientes al final de los ochenta en el campus de la **CMU**, se volvió tan grande que los retrasos en la red y los fallos del servidor ocurrían en cualquier lugar cada día provocando numerosas molestias.

Se vió en la sección anterior, que **Coda** cachea toda la información que necesita en el cliente para proveer acceso a los datos. Cuando se realiza una actualización del sistema de ficheros esta debe propagarse al servidor. En el modo conectado normal, esas actualizaciones se propagan de manera síncrona al servidor, esto es, cuando la actualización es completada en el cliente, esta también se ha hecho en el servidor. Si un servidor no está disponible o si las conexiones de red entre el cliente y el servidor fallan, la operación recibirá un error de **time-out** y fallará. Algunas veces no se podrá hacer nada. Por ejemplo, intentar obtener un archivo que no está en la caché desde los servidores sin conexión de red es imposible. En esos casos el error se enviará al programa que realizó la llamada. Sin embargo, el **time-out** puede manejarse a menudo de una forma elegante, como se detalla a continuación.

Para soportar ordenadores desconectados u operar ante la presencia de fallos de red, **Venus** no enviará informes de los fallos al usuario cuando una actualización obtenga un **time-out**. En lugar de eso, **Venus** se da cuenta de que el servidor en cuestión no está disponible, y que la actualización deberá ser registrada en el servidor. Durante la desconexión, todas las actualizaciones se almacenan en el **CML**, el registro de modificaciones del cliente, que es salvado frecuentemente en el disco. El usuario no se da cuenta de que **Coda** ha cambiado a modo desconectado. Una vez que se han reconectado los servidores, **Venus** reintegrará el **CML**: pedirá a los servidores repetir las actualizaciones del sistema de ficheros local en el servidor, llevando de ese modo el servidor al estado actual. Además el **CML** está optimizado, por ejemplo, el ignora las operaciones sobre un fichero que ha sido creado y posteriormente borrado.

Hay otros dos resultados de profunda importancia para el funcionamiento desconectado. El primero que hay es el concepto de *acumulación de archivos* . Desde que **Venus** no puede servir un fallo en el caché durante una desconexión, podría ser apropiado que mantuviera ciertos archivos en la caché actualizados, pidiendo frecuentemente al servidor que le envíe las

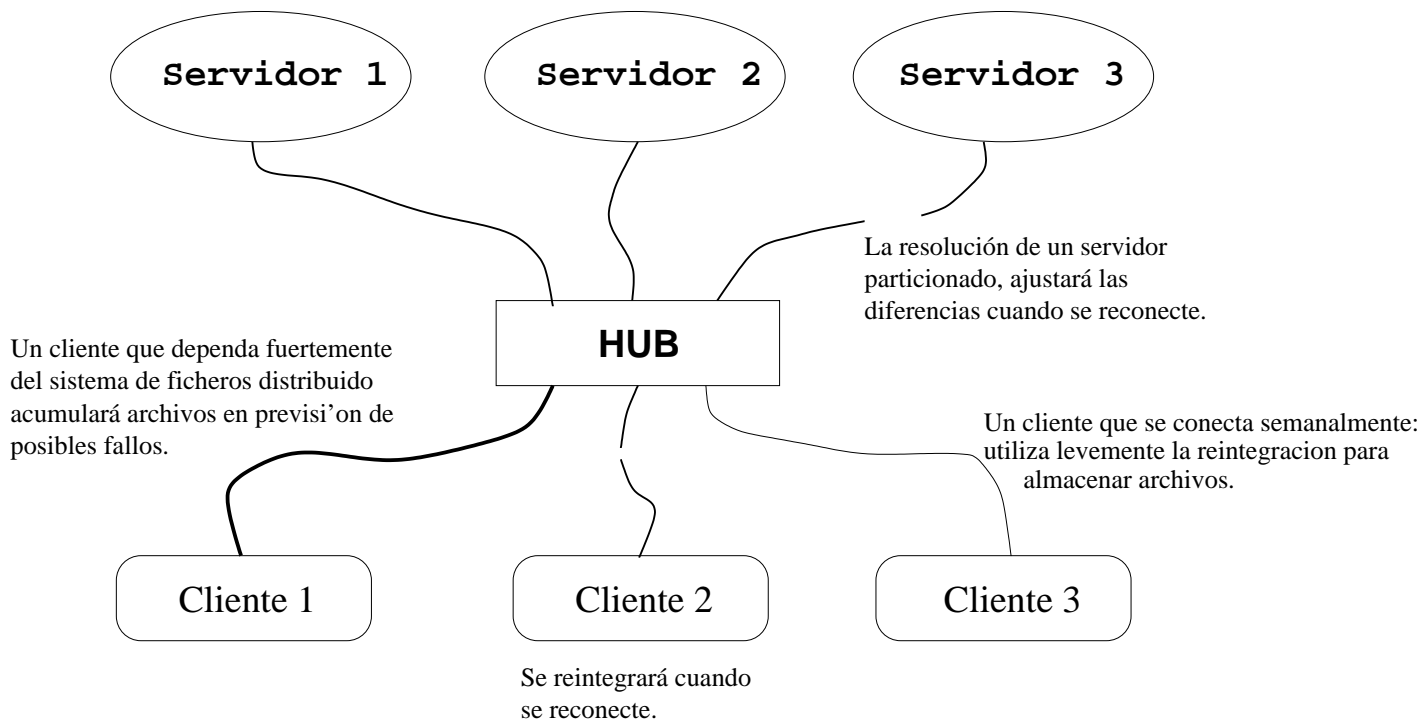


Figura 2: Clientes y servidores en distintos estados.

últimas actualizaciones si es necesario. Esos archivos importantes están en *las bases de datos de acumulación* de los usuarios (que pueden ser construidas automáticamente "*espiando*" en los accesos a archivo de los usuarios). Actualizar los archivos *acumulados* se denomina el *recorrido de acumulación*. En la práctica nuestros ordenadores acumulan enormes cantidades de software del sistema, como los binarios y bibliotecas del sistema de ventanas **X11** o **Wabi** y **Microsoft Office**.

El segundo resultado es que durante la reintegración podría aparecer que durante la desconexión otro cliente ha modificado el archivo varias veces y debe el archivo debe transportarse al servidor. Esto se denomina un conflicto **local/global** y necesita repararse. Algunas veces, las reparaciones pueden realizarse automáticamente por las aplicaciones específicas preparadas para ello (que saben que si un cliente ha insertado una cita para el lunes en el archivo del calendario y otro cliente ha insertado una cita para el martes, ésto no ha creado un conflicto irresoluble). Algunas veces, pero muy infrecuentes, se requiere la intervención de una persona para reparar el conflicto.

Así el viernes alguien deja la oficina con bastante código fuente acumulado en el portatil. Después de estar codificando en una cabaña en las montañas, el cruel retorno a la oficina el lunes (diez días después desde luego), comienza con la reintegración de las actualizaciones realizadas durante el fin de semana. La computación ambulante ha nacido.

### 5.3 Volúmenes, Servidores y Replicación de Servidores.

En muchos sistemas de ficheros de red el servidor disfruta de una estructura estándar de archivos y exporta un directorio a los clientes. Tal directorio de archivos en el servidor puede ser montado en el cliente y se denomina un **recurso compartido** en la jerga de *Windows* y un **sistema de ficheros en red** en el mundo *Unix*. Para muchos de esos sistemas no es práctico montar más sistemas distribuidos dentro de los volúmenes de red montados. Se pone especial cuidado y atención en la distribución de las particiones del servidor, sus directorios, y sus recursos compartidos. La organización de **Coda** (y **AFS**) difiere de manera sustancial.

Los archivos en los servidores **Coda** no se almacenan en un sistema de ficheros tradicional. La organización es como sigue. Las particiones en los ordenadores servidores **Coda** pueden ponerse como disponibles al servidor de ficheros. Esas particiones contendrán archivos que son agrupados en volúmenes. Cada volumen tiene una estructura de directorios como en un sistema de ficheros, esto es: un directorio raíz del volumen y un árbol bajo el. Un volumen completo es mucho más pequeño que una partición, pero mucho mayor que un simple directorio y es una unidad lógica de archivos. Por ejemplo, el directorio *home* de un usuario podría ser normalmente un único volumen **Coda** y de manera similar los fuentes de **Coda** podrían residir en un único volumen. Típicamente un único servidor puede tener varios cientos de volúmenes, quizás con un tamaño medio de aproximadamente 10Mb. Un volumen es una cantidad manejable de datos de archivos que es una unidad muy natural desde la perspectiva de la administración del sistema y se ha demostrado bastante flexible.

**Coda** mantiene información sobre los volúmenes y los directorios, las listas de control de acceso, y los atributos de los ficheros en particiones *crudas* (sin formatear). Estas son accedidas a través de un registro basado en el paquete de memoria virtual recuperable (**RVM**) para una mayor velocidad y consistencia. Sólo los datos de los ficheros residen en los archivos de las particiones del servidor. El **RVM** ha sido construido para que soporte transacciones, esto significa que en caso de que el servidor se estropee el sistema puede restaurarse a un estado consistente sin mucho esfuerzo.

Un volumen tiene un nombre y una identificación y es posible montar un volumen en cualquier lugar bajo */coda*. Por ejemplo, para montar el volumen **u.braam** en */coda/usr/braam*, se utiliza el comando **cfs makemount u.braam /coda/usr/braam**. **Coda** no permite que los puntos de anclaje sean directorios existentes, ya que se creará un nuevo directorio como parte del proceso de anclaje (o montaje). Esto elimina la confusión que se puede crear al montar sistemas de ficheros Unix sobre directorios existentes. Mientras parece muy similar a la tradición en *Macintosh* y *Windows* de crear "unidades de disco y volúmenes" de red, la diferencia principal es que el punto de anclaje es invisible al cliente: este aparece como un directorio ordinario bajo */coda*. Un único volumen disfruta del privilegio de ser un volumen raíz, es el volumen que es montado en */coda* en el momento de la inicialización.

**Coda** identifica un archivo por un vector de tres enteros de **32 bits** denominado **Fid**: consiste de una identificación de volumen (**VolumeId**), una identificación de inodo (**VnodeId**) y un identificador único. El identificador de volumen identifica el volumen en que reside el archivo. El identificador de inodo es el **número de inodo** del archivo. El vector **Fid** es único en un *cluster* de servidores **Coda**.

**Coda** tiene servidores de replicación de lectura/escritura, por ejemplo, un grupo de servidores puede ofrecer datos de archivos a clientes y generalmente las actualizaciones se hacen a todos los servidores del grupo. La ventaja de este sistema es una mayor disponibilidad de los datos: si un servidor falla, otro tomará el control sin notificar al cliente el fallo. Los volúmenes pueden almacenarse en un grupo de servidores denominados **VSG** (Volume Storage Group, Grupo de almacenamiento de Volúmenes).

Para volúmenes replicados el **VolumeId** es un **VolumeId** replicado. La identificación del volumen replicado lleva ligada un **VSG** y un volumen local en cada uno de los miembros.

El **VSG** es una lista de servidores que mantiene una copia del volumen replicado.

El volumen local para cada servidor define una partición y un **VolumeId** que mantiene los archivos y metadatos en ese servidor.

Cuando **Venus** desea acceder a un objeto en los servidores, primero necesita encontrar la información del volumen **VolumeInfo** para el volumen que contiene el archivo. Esta información contiene la lista de los servidores y los identificadores de los volúmenes locales de cada servidor cuyo volumen es conocido. Para los archivos la comunicación con los servidores de un **VSG** es **read-one, write-many** (*leer-uno, escribir-muchos*), esto es: lee el archivo de un servidor único en el **VSG** y propaga los datos a todos los miembros del **VSG** disponibles, el **AVSG**. **Coda** puede emplear **multicast RPCs**, para que así escribir muchas (**write-many**) actualizaciones no sea una penalización para las prestaciones del servidor.

La sobrecarga de la primera vez que se tiene que obtener la información del volumen es engañosa también. Aunque hay un tiempo para la búsqueda de la información del volumen, los subsiguientes accesos a los archivos disfrutan de un camino mucho más corto desde que el acceso desde la raíz del volumen.

La replicación de servidores como el funcionamiento desconectado tiene dos aspectos que necesitan presentarse: la resolución y la reparación. Algunos servidores en el **VSG** pueden particionarse a través de la red o por fallos de servidores. En ese caso el **AVSG** algunos objetos será mas pequeño estrictamente que el **VSG**. Las actualizaciones no se pueden propagar a todos los servidores, sólo los miembros del **AVSG**, de esta forma introducen conflictos globales.

Antes de obtener un objeto o sus atributos, **Venus** requerirá la versión marcada de todos los

servidores disponibles. Si detecta que algún servidor no tiene la última copia de los archivos, inicia un proceso de resolución que tratará de resolver automáticamente esa diferencia. Si esto falla, un usuario deberá repararlo manualmente. La resolución, iniciada por el cliente, es manejada enteramente por los servidores.

Los servidores de replicación y resolución son maravillosos. Si se sufren fallos de disco de vez en cuando en algunos servidores, para reparar el servidor, todo lo que se necesita es instalar un nuevo disco y decirle a **Coda** que lo resuelva. El sistema de resolución construye el nuevo disco actualizado basándose en el resto de servidores.

#### 5.4 Coda en acción.

**Coda** está constantemente activo en el **CMU**. Muchos clientes lo usan para el trabajo de desarrollo (de **Coda**), como una propuesta general para el sistema de archivos y para las aplicaciones desconectadas específicas. En los dos escenarios siguientes están explotando las características de coda con mucho éxito. En el **CMU** se han comprado varias licencias de *Wabi* (un emulador de windows para Unix) y algún software de windows asociado. *Wabi* permite a los usuarios ejecutar *Ms. Powerpoint* y otros programas. Se ha almacenado *Wabi*, *Windows* y *Ms. Office* en **Coda**, y son compartidos por los clientes. Desde luego, los archivos **.ini** con las preferencias de un usuario determinado son particulares, pero muchas bibliotecas y aplicaciones no lo son. Mediante el uso de la *acumulación de ficheros*, se puede continuar usando el software para presentaciones realizadas en ordenadores portátiles. Esto se hace frecuentemente en las conferencias.

Se ha usado durante años sin pérdidas de datos de los usuarios. Algunas veces los discos de los servidores han fallado, pero como todos los volúmenes están replicados, se reemplaza el disco con uno vacío y se invoca al mecanismo de resolución para actualizar el servidor reparado. Todo lo que se necesita para hacer esto es escribir **ls -IR** en el árbol de directorios afectado cuando se ha colocado el nuevo disco. La ausencia del archivo en el servidor reparado será notificada y la **resolución** transportará los archivos de los servidores en buen estado al servidor reparado.

Hay un número de obligadas aplicaciones futuras donde coda podría proveer significantes beneficios.

Los *mirrors* de FTP deberían ser clientes **Coda**. Por ejemplo, tomemos **ftp.redhat.com**, que tiene varios *mirrors*. Cada mirror activa un programa en **Perl** que recorre el árbol de directorios completo de **RedHat** para ver qué se ha actualizado y obtenerlo, aunque esto no sea necesario. Por el contrario, si **RedHat** almacenara su área FTP en **Coda**, los *mirrors* se convertirían en clientes, (sólo **RedHat** tendría permiso de escritura) y cuando se actualizara un paquete, los servidores **Coda** notificarían a los *mirrors* que el archivo ha cambiado. El

mirror obtendría este paquete de los servidores, pero sólo la próxima vez que alguien intente obtener ese paquete.

Los servidores de replicación de WWW deberían ser clientes **Coda**. Muchos proveedores de servicios de internet están luchando con sólo unos pocos servidores de replicación de WWW. Tienen demasiados accesos para usar un único servidor **http**. Usar NFS para distribuir los documentos que se van a servir, se ha mostrado problemático debido a problemas con las prestaciones, así que la copia manual de archivos a servidores individuales se realiza de manera frecuente. **Coda** podría resolver esto, ya que cada servidor podría ser un cliente **Coda** y mantener los datos en su caché. Esto provee velocidades de acceso similares a las de un disco local. Si se combina esto con que los clientes del **ISP** actualizan su información **offline**, se obtiene una buena aplicación para clientes móviles también.

Los ordenadores de una red podrían utilizar **Coda** como una caché usada para mejorar las prestaciones de una manera drástica. Las actualizaciones en los ordenadores de la red podrían ser hechas automáticamente cuando se convierten en servidores disponibles, y gran parte del ordenador, podría funcionar sin tráfico de red, incluso después de un rearranque.

Los esfuerzos actuales en **Coda** son en su mayoría para mejorar la calidad de **Coda**. Los ásperos límites, que inevitablemente aparecen con los sistemas en investigación son lentamente suavizados. Se añadirá a **Coda** una caché de tipo **write-back** que le permitirá funcionar mucho más rápido. El funcionamiento desconectado es una forma extrema de escritura **write-back** en el caché, y se está aprovechando el conocimiento de esos mecanismos para realizar escrituras **write-back** durante el funcionamiento conectado. También se está añadiendo soporte para **Kerberos**. Los protocolos de red que soportan **Coda** se están haciendo tan sencillos como es posible. Sería interesante tener celdas que permitieran a los clientes conectarse a más de un *cluster* **Coda** simultáneamente. Las distintas traducciones que se están realizando para distintas plataformas permitirán a muchos sistemas distintos usar **Coda**.

## 6 Bibliografía.

### The Coda HOWTO

- Peter Braam, Robert Baron, Jan Harkes, Marc Schnieder, v1.01, 16 Jan, 2000
- <http://www.coda.cs.cmu.edu/doc/html/coda-howto.html>

### The Coda Distributed File System

- Peter Braam (School of Computer Science, Carnegie Mellon University)

- <http://www.coda.cs.cmu.edu/ljpaper/lj.html>

**Sistemas operativos distribuidos.**

- Andrew S. Tanenbaum.
- Primera edición. Prentice Hall, 1995.