

Sistemas Distribuidos

4

Sistemas de ficheros distribuidos

Índice

- Introducción
- Estructura de un SFD
- Resolución de nombres
- Acceso a los datos
- Gestión de cache
- Gestión de cerrojos
- Estudio de ejemplos: NFS, AFS y Coda
- Sistemas de ficheros paralelos
 - *General Parallel File System* (GPFS)
 - *Google File System* (GFS)

Conceptos básicos

- Sistema de ficheros distribuido (SFD)
 - Sistema de ficheros para sistema distribuido
 - Gestiona distintos dispositivos en diferentes nodos ofreciendo a usuarios la misma visión que un SF centralizado
 - Permite que usuarios compartan información de forma transparente
 - Misma visión desde cualquier máquina
- Numerosos aspectos similares a SF centralizados
- Algunos aspectos específicos como por ejemplo:
 - Traducción de nombres involucra a varios nodos
 - Uso de cache afecta a múltiples nodos
 - Aspectos de tolerancia a fallos

Características deseables del SFD

- Aplicables las correspondientes al SD global:
 - Transparencia, fiabilidad, rendimiento, escalabilidad, seguridad.
- Específicamente:
 - Espacio de nombres único
 - Soporte de migración de ficheros
 - Soporte para replicación
 - Capacidad para operar en sistemas “desconectados”
 - Una red “partida” o un cliente que usa un sistema portátil
 - Soporte de heterogeneidad en hardware y S.O.
 - Integración de nuevos esquemas de almacenamiento (SAN)
 - Paralelismo en acceso a datos de un fichero

Sistemas Distribuidos

Estructura del SFD

- Cliente (nodo con aplicación) – Servidor (nodo con disco)
- ¿Cómo repartir funcionalidad de SF entre cliente y servidor?
- Arquitectura “tradicional”
 - servidor: proporciona acceso a ficheros almacenados en sus discos
 - cliente: pasarela entre aplicación y servidor
 - con más o menos funcionalidad (clientes *fat o thin*)
- Arquitectura “alternativa”
 - servidor: proporciona acceso a bloques de disco
 - cliente: toda la funcionalidad del SF
 - Utilizada en sistemas de ficheros para *clusters*
 - se estudiará más adelante

Sistemas Distribuidos
5

Fernando Pérez Costoya

Arquitectura del SFD

- Solución basada en arquitectura tradicional parece sencilla:
 - SF convencional en servidor
 - Exporta servicios locales para abrir, cerrar, leer, escribir, cerrojos,...
- ¿Asunto zanjado? No todo está resuelto:
 - Resolución de nombre de fichero:
 - Cliente y varios servidores involucrados. ¿Cómo se reparten trabajo?
 - Acceso a los datos:
 - ¿se transfiere sólo lo pedido? ¿más cantidad? ¿todo el fichero?
 - Uso de cache en el cliente. Coherencia entre múltiples caches.
 - Gestión de cerrojos:
 - ¿Qué hacer si se cae un cliente en posesión de un cerrojo?
 - Otros: migración, replicación, heterogeneidad, ...

Sistemas Distribuidos
6

Fernando Pérez Costoya

Operaciones sobre los ficheros

- Apertura del fichero: Traducción del nombre
 - Gestión de nombres
- Lecturas/escrituras sobre el fichero
 - Acceso a datos
 - Uso de cache
- Establecimiento de cerrojos sobre el fichero

Sistemas Distribuidos
7

Fernando Pérez Costoya

Gestión de nombres

- Similar a SF convencionales:
 - Espacio de nombres jerárquico basado en directorios
 - Esquema de nombres con dos niveles:
 - Nombres de usuario (*pathname*) y Nombres internos
 - Directorio: Relaciona nombres de usuario con internos
- Nombres de usuario
 - Deben proporcionar transparencia de la posición
 - Nombre no debe incluir identificación del nodo donde está
triqui.fi.upm.es:/home/fichero.txt
- Nombres internos
 - Identificador único de fichero (UFID) utilizado por el sistema
 - Puede ser una extensión del usado en SF convencionales.
 - Por ejemplo:
id. de máquina + id. disco + id. partición + id. inodo

Sistemas Distribuidos
8

Fernando Pérez Costoya

Sistemas Distribuidos

Espacio de nombres

- Similar a SF convencionales:
 - Espacio de nombres dividido en volúmenes (o particiones, o ...)
 - Cada volumen gestionado por un servidor
 - Espacio único mediante composición de volúmenes:
 - Extensión distribuida de operación de montaje de UNIX
- Alternativas en la composición:
 - Montar sistema de ficheros remoto sobre la jerarquía local (NFS)
 - Montaje en cliente: información de montaje se almacena en cliente
 - Espacio de nombres diferente en cada máquina
 - Único espacio de nombres en todas las máquinas (AFS)
 - Montaje en servidor: información de montaje se almacena en servidor
 - Espacio de nombres común para el SD

Sistemas Distribuidos
9

Fernando Pérez Costoya

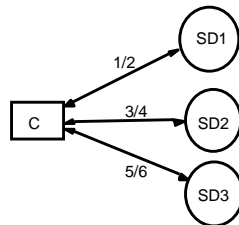
Resolución de nombres

- Traducir una ruta que se extiende por varios servidores
- ¿Quién busca cada componente de la ruta?
 - cliente: solicita contenido del directorio al servidor y busca (AFS)
 - servidor: realiza parte de la búsqueda que le concierne
- Alternativas en la resolución dirigida por los servidores
 - iterativa, transitiva y recursiva
- "Cache de nombres" en clientes
 - Almacén de relaciones entre rutas y nombres internos
 - También existe en SF convencional
 - Evita repetir proceso de resolución
 - Operación más rápida y menor consumo de red (*escalabilidad*)
 - Necesidad de coherencia
 - Fichero borrado y nombre interno reutilizado
 - Uso de contador de versión del inodo
 - id. de máquina + id. disco + id. partición + id. inodo + n° versión*

Sistemas Distribuidos
10

Fernando Pérez Costoya

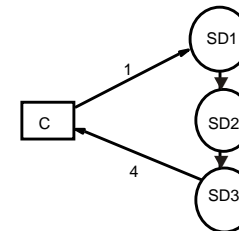
Resolución iterativa



Sistemas Distribuidos
11

Fernando Pérez Costoya

Resolución transitiva

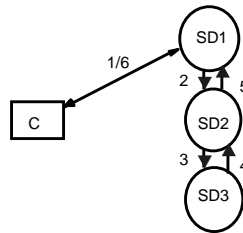


Sistemas Distribuidos
12

Fernando Pérez Costoya

Sistemas Distribuidos

Resolución recursiva



Sistemas Distribuidos
13

Fernando Pérez Costoya

Localización de ficheros

- Resolución obtiene nombre interno (UFID)
- Uso de UFID con dir. máquina donde fichero está localizado
 - No proporciona, en principio, independencia de la posición
 - Nombre de fichero cambia cuando éste migra
- Uso de UFID que no contenga información de máquina:
 - Por ejemplo (AFS):
 - id. único de volumen + id. inodo + n° versión*
 - Permite migración de volúmenes
 - Requiere esquema de localización: Volumen → Máquina
- Posibles esquemas de localización:
 - Tablas que mantengan la información de ubicación (AFS)
 - Uso de *broadcast* para localizar nodo
- Uso de "cache de localizaciones" en clientes

Sistemas Distribuidos
14

Fernando Pérez Costoya

Acceso a datos del fichero

- Una vez abierto el fichero, se tiene info. para acceder al mismo
- Aspectos de diseño vinculados con acceso a datos:
 - ¿Qué se garantiza ante accesos concurrentes?
 - Semántica de uso concurrente
 - ¿Qué información se transfiere entre cliente y servidor?
 - Modelo de acceso
 - ¿Qué info. se guarda en cache y cómo se gestiona?
 - Gestión de cache

Sistemas Distribuidos
15

Fernando Pérez Costoya

Semánticas de uso concurrente

- Sesión: serie de accesos que realiza cliente entre *open* y *close*
- La semántica especifica el efecto de varios procesos accediendo de forma simultánea al mismo fichero
- Semántica UNIX
 - Una lectura ve los efectos de todas las escrituras previas
 - El efecto de dos escrituras sucesivas es el de la última
- Semántica de sesión (AFS)
 - Cambios a fichero abierto, visibles sólo en nodo que lo modificó
 - Una vez cerrado, cambios visibles sólo en sesiones posteriores
 - Múltiples imágenes simultáneas del fichero
 - Dos sesiones sobre mismo fichero que terminan concurrentemente:
 - La última deja el resultado final
 - No adecuada para procesos con acceso concurrente a un fichero

Sistemas Distribuidos
16

Fernando Pérez Costoya

Modelo de acceso

- Modelo carga/descarga
 - Transferencias completas del fichero
 - Localmente se almacena en memoria o discos locales
 - Normalmente utiliza semántica de sesión
 - Eficiencia en las transferencias
 - Llamada *open* con mucha latencia
- Modelo de servicio remoto
 - Servidor debe proporcionar todas las operaciones sobre el fichero
 - Acceso por bloques
 - Modelo cliente/servidor

Modelo carga/descarga

- Correspondencia petic. de aplicación y mens. de protocolo:
 - *open* → mensaje de descarga (*download*)
 - se realiza traducción y servidor envía fichero completo
 - cliente almacena fichero en cache local
 - *read/write/seek* → no implica mensajes de protocolo
 - lecturas y escrituras sobre copia local
 - *close* → mensaje de carga (*upload*)
 - si se ha modificado, se envía fichero completo al servidor

Modelo de serv. remoto: con estado

- Correspondencia petic. de aplicación y mens. de protocolo:
 - *open* → mensaje de apertura
 - se realiza traducción
 - servidor habilita zona de memoria para info. de la sesión
 - retorna id. específico para esa sesión
 - *read/write/seek* → mensaje de protocolo correspondiente
 - mensaje incluye id. sesión
 - mensaje lectura y escritura incluye tamaño pero no posición
 - uso de información sobre la sesión almacenada en servidor (posición)
 - *close* → mensaje de cierre
 - servidor libera zona de memoria de la sesión
- Ventajas servicio con estado (frente a sin estado):
 - Mensajes más pequeños, posibilidad de realizar políticas "inteligentes" en servidor (p.ej. lectura anticipada), procesamiento de peticiones posiblemente un poco más eficiente

Modelo de serv. remoto: sin estado

- Correspondencia petic. de aplicación y mens. de protocolo:
 - *open* → mensaje de apertura
 - se realiza traducción
 - servidor **no** habilita zona de memoria para info. de sesión (cliente sí)
 - retorna id. interno del fichero UFID
 - *read/write* → mensaje de protocolo correspondiente
 - mensaje autocontenido
 - mensaje incluye UFID, tamaño y posición
 - *seek* → no implica mensaje de protocolo
 - *close* → no implica mensaje de protocolo
 - cliente libera zona de memoria de la sesión
- Ventajas servicio sin estado (frente a con estado):
 - Tolerancia a fallos ante re arranque del servidor, posiblemente menos mensajes, no hay gastos de recursos en servidor por cada cliente (*escalabilidad*)

Gestión de cache

- El empleo de cache permite mejorar el rendimiento
- Caches en múltiples niveles de un SD:
 - Cache en los servidores
 - Reducen los accesos a disco
 - Cache en los clientes
 - Reduce el tráfico por la red
 - Reduce la carga en los servidores
 - Puede situarse en discos locales (no permite nodos sin disco)
 - Más capacidad pero más lento
 - No volátil, facilita la recuperación
 - y/o en memoria principal
 - Menor capacidad pero más rápido
 - Memoria volátil

Uso de cache en clientes

- Empleo de cache de datos en clientes
 - Mejora rendimiento y capacidad de crecimiento
 - Introduce problemas de coherencia
- Otros tipos de cache
 - Cache de nombres
 - Cache de metadatos del sistema de ficheros
- Políticas de gestión de cache de datos:
 - Política de actualización
 - Política de coherencia
- Enfoque alternativo: Cache colaborativa (xFS)
 - Si bloque solicitado está en la cache de otro cliente, se copia de ésta

Política de actualización

- Escritura inmediata (*write through*)
 - Buena fiabilidad
 - Las escrituras son más lentas
 - Mayor fragmentación en la información transferida por la red
- Escritura diferida (*delayed write*)
 - Escrituras más rápidas
 - Se reduce el tráfico en la red
 - Los datos pueden borrarse antes de ser enviados al servidor
 - Menor fiabilidad
 - ¿Cuándo volcar los datos?
 - Volcado periódico
 - Volcado al cerrar (*Write-on-close*)

Coherencia de cache

- El uso de cache en clientes produce problema de coherencia
 - ¿es coherente una copia en cache con el dato en el servidor?
- Estrategia de validación iniciada por el cliente
 - cliente contacta con servidor para determinar validez
 - en cada acceso, al abrir el fichero o periódicamente
- Estrategia de validación iniciada por el servidor
 - servidor avisa a cliente (*callback*) al detectar que su copia es inválida
 - generalmente se usa *write-invalidate* (no *write-update*)
 - servidor almacena por cada cliente info. sobre qué ficheros guarda
 - implica un servicio con estado

C. de cache: semántica de sesión

- Validación iniciada por el cliente (usada en AFS versión 1):
 - En apertura se contacta con servidor enviando nº de versión (o fecha modificación) del fichero almacenado en cache local (si lo hay)
 - Servidor comprueba si corresponde con versión actual:
 - En caso contrario, se envía la nueva copia
- Validación iniciada por el servidor (usada en AFS versión 2):
 - Si hay copia en cache local, en apertura no se contacta con servidor
 - Servidor almacena información de qué clientes tienen copia local
 - Cuando cliente vuelca nueva versión del fichero al servidor:
 - servidor envía invalidaciones a clientes con copia
 - Disminuye nº de mensajes entre cliente y servidor
 - Mejor rendimiento y *escalabilidad*
 - Dificultad en la gestión de *callbacks*
 - No encajan fácilmente en modelo cliente-servidor clásico

C. de cache: semántica UNIX

- Validación iniciada por el cliente
 - Inaplicable
 - Hay que contactar con servidor en cada acceso para validar info.
- Validación iniciada por el servidor. 2 ejemplos de protocolos:
 - *Prot1*: control en la apertura con desactivación de cache
 - Basado en Sprite: SOD desarrollado en Berkeley en los 80
 - *Prot2*: uso de *tokens*
 - Basado en DFS, sistema de ficheros distribuido de DCE (*Open Group*)

C. de cache: semántica UNIX. *Prot1*

- Servidor guarda info. de qué clientes tienen abierto un fichero
- Si acceso concurrente conflictivo (1 escritor + otro(s) cliente(s))
 - se anula cache y se usa acceso remoto en nodos implicados
- En *open* cliente contacta con servidor especificando:
 - modo de acceso + nº de versión de copia en cache (si la hay)
 - Si no hay conflicto de acceso:
 - Si versión del cliente en cache es más antigua, se indica que la invalide
 - Si la petición produce un conflicto de acceso:
 - Se le envía a los clientes con el fichero abierto una orden de invalidación y desactivación de la cache para ese fichero
 - Si era un escritor se le pide un volcado previo
 - Se indica al cliente que invalide y desactive la cache para ese fichero
 - Si la petición se encuentra que ya hay conflicto
 - Se indica al cliente que invalide y desactive la cache para ese fichero

C. de cache: semántica UNIX. *Prot2*

- Para realizar operación se requiere *token* correspondiente
 - *Token* (de lectura o escritura) asociado a un rango de bytes
- Si cliente solicita operación y no está presente *token* requerido en su nodo, se solicita al servidor de ficheros
- Para una zona de un fichero, el servidor puede generar múltiples *tokens* de lectura pero sólo uno de escritura
- Si existen múltiples *tokens* de lectura y llega solicitud de escritura, servidor reclama los *tokens*
 - Cliente devuelve *token* e invalida bloques de cache afectados
 - Cuando todos devueltos, servidor manda *token* de escritura
- Si hay un *token* de escritura y llega solicitud de lectura o escritura, servidor reclama el *token*.
 - Cliente vuelca e invalida bloques de cache afectados

Sistemas Distribuidos

Servicio con estado basado en *leases*

- Semántica UNIX requiere servicio con estado
 - ¿Cómo lograr servicio con estado pero con buena recuperación?
- *Lease*: concesión con plazo de expiración
 - Necesita renovarse
 - Aplicable a otros servicios además de la coherencia
- *Prot2* ampliado: *token* tiene un plazo de expiración
 - Pasado el plazo cliente considera que *token* ya no es válido
 - Tiene que volver a solicitarlo
 - Permite tener servidor con estado pero fácil recuperación:
 - Cuando reanuncia servidor no entrega *tokens* hasta que haya pasado plazo de expiración con lo que todos los *tokens* están caducados

Sistemas Distribuidos
29

Fernando Pérez Costoya

Gestión de cerrojos

- SFD ofrecen cerrojos de lectura/escritura
 - múltiples lectores y un solo escritor
- Peticiones *lock/unlock* generan mensajes correspondientes
 - *lock*: si factible retorna OK; sino no responde
 - *unlock*: envía a OK a cliente(s) en espera
- Requiere un servicio con estado:
 - servidor almacena qué cliente(s) tienen un cerrojo de un fichero y cuáles están en espera
- Problema: cliente con cerrojo puede caerse
 - Solución habitual: uso de *leases*
 - Cliente con cerrojo debe renovarlo periódicamente

Sistemas Distribuidos
30

Fernando Pérez Costoya

Network File System (NFS) de Sun

- Especificación de un protocolo para acceso a ficheros remotos
- Estándar diseñado para entornos heterogéneos
 - Versión 3: RFC-1813 (descrita en esta presentación)
 - Versión 4: RFC-3010 (última versión; cambios en arquitectura)
- Independencia gracias al uso de RPC/XDR de ONC
- Seguridad basada en "RPC segura"
- Compartición: máquina monta directorio remoto en SF local
 - Espacio de nombres es diferente en cada máquina
- No da soporte a migración ni replicación
- Comprende dos protocolos: montaje y acceso a ficheros

Sistemas Distribuidos
31

Fernando Pérez Costoya

Protocolo de montaje

- Establece una conexión lógica entre el servidor y el cliente
- Cada máquina incluye una "lista de exportación"
 - qué "árboles" exporta y quién puede montarlos
- Petición de montaje incluye máquina y directorio remotos
 - Se convierte en RPC al servidor de montaje remoto
 - Si permiso en lista, devuelve un identificador "opaco" (*handle*)
 - Cliente no conoce su estructura interna
- La operación de montaje sólo afecta al cliente no al servidor
 - se permiten montajes NFS anidados, pero no "transitivos"
- Aspectos proporcionados por algunas implementaciones:
 - montajes *hard* o *soft*: en montaje, si servidor no responde...
 - espera ilimitada (*hard*) o plazo máximo de espera (*soft*)
 - automontaje: no solicita montaje hasta acceso a ficheros

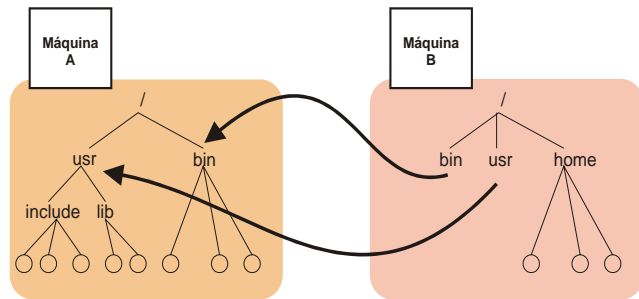
Sistemas Distribuidos
32

Fernando Pérez Costoya

Sistemas Distribuidos

Ejemplo de montaje en NFS

- La máquina A exporta `/usr` y `/bin`
- En la máquina B:
 - `mount máquinaA:/usr /usr`

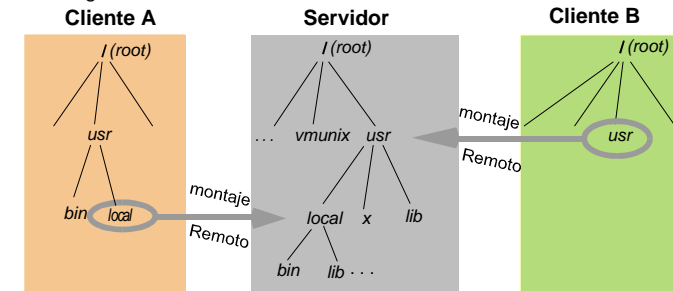


Sistemas Distribuidos
33

Fernando Pérez Costoya

Ejemplo de montaje en NFS

- Imagen diferente del sistema de ficheros



Sistemas Distribuidos
34

Fernando Pérez Costoya

Protocolo NFS

- Ofrece RPCs para realizar operaciones sobre ficheros remotos
 - Búsqueda de un fichero en un directorio (LOOKUP)
 - Lectura de entradas de directorio
 - Manipulación de enlaces y directorios
 - Acceso a los atributos de un fichero
 - Lectura y escritura. En vers. 2 escritura síncrona en disco de servidor
 - Versión 3 permite asíncrona (COMMIT fuerza escritura en disco)
- Servidores NFS sin estado (no en versión 4)
 - Operaciones autocontenidas
- OPEN reemplazado por LOOKUP (no hay CLOSE)
 - traducción iterativa componente a componente
 - LOOKUP(*handle* de directorio, fichero) → *handle* de fichero
- El protocolo no ofrece mecanismos de control de concurrencia
 - Procolo independiente de NFS: *Network Lock Manager*

Sistemas Distribuidos
35

Fernando Pérez Costoya

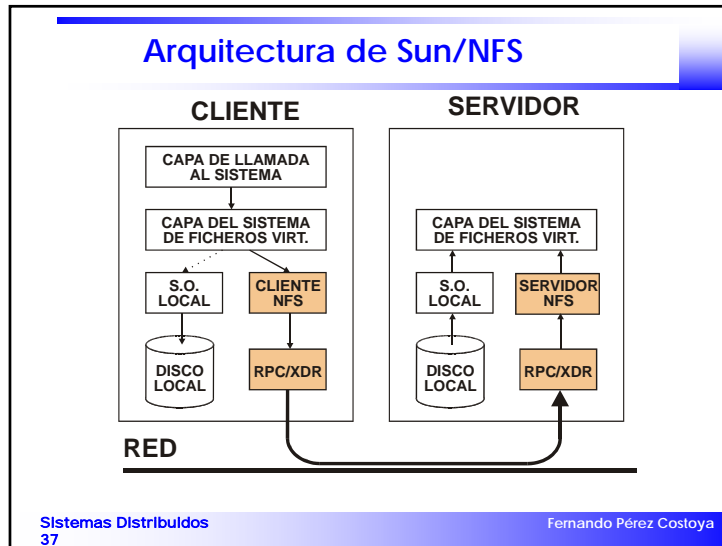
Implementación Sun/NFS

- Arquitectura basada en sistema de ficheros virtual (VFS)
- *Vnode* apunta a un nodo *ilocal* o a uno remoto (*Rnode*)
- Cada *Rnode* contiene *handle* del fichero remoto
- Contenido del *handle* depende de sistema remoto
- En sistemas UNIX se usa un *handle* con tres campos:
 - id. del sistema de ficheros
 - número de inodo
 - número del versión del inodo (se incrementa en cada reutilización)
- En montaje se obtiene *handle* de la raíz del subárbol montado
- Posteriores operaciones *lookup* obtienen sucesivos *handles*

Sistemas Distribuidos
36

Fernando Pérez Costoya

Sistemas Distribuidos



Acceso a ficheros en Sun/NFS

- Las transferencias se realizan en bloques de 8 KB
- Los bloques se almacenan en la cache de los clientes
- Los clientes realizan lecturas adelantadas de un bloque
- Las escrituras se realizan localmente.
- Los bloques se envían al servidor cuando se completan o se cierra el fichero
- Cache del servidor:
 - escritura síncrona o asíncrona según lo indicado por el cliente
- 3 tipos de cache en el cliente:
 - cache de nombres para acelerar las traducciones
 - cache de atributos de ficheros y directorios (fechas, dueño, ...)
 - cache de bloques de ficheros y directorios

Sistemas Distribuidos 38 Fernando Pérez Costoya

Coherencia de cache en Sun/NFS

- No asegura ninguna semántica
- Validación dirigida por el cliente:
 - Toda operación sobre un fichero devuelve sus atributos
 - Si los atributos indican que el fichero se ha modificado
 - se invalidan los datos del fichero en cache de bloques
 - Entradas de cache de bloques y atributos tienen un tiempo de vida
 - Si no se acceden es ese periodo se descartan
 - Valores típicos:
 - 3 segundos para ficheros
 - 30 para directorios

Sistemas Distribuidos 39 Fernando Pérez Costoya

Novedades de la versión 4 de NFS

- Servicio con estado basado en *leases* (hay OPEN y CLOSE)
- Diseñado para ser usado en Internet
- Integra protocolo de montaje y de cerrojos
 - Un solo puerto fijo (2049): facilita atravesar cortafuegos
- Empaquetamiento de operaciones:
 - Una llamada RPC (COMPOUND) con múltiples operaciones
- Operación LOOKUP puede resolver el camino completo
 - Montajes en el servidor visibles por el cliente
- Uso de listas de control de acceso (ACL)
- Atributos del fichero incluye un mecanismo de extensibilidad

Sistemas Distribuidos 40 Fernando Pérez Costoya

Sistemas Distribuidos

Andrew File System (AFS)

- SFD desarrollado en Carnegie- Mellon (desde 1983)
 - Se presenta la versión AFS-2
- Actualmente producto de Transarc (incluida en IBM)
 - OpenAFS: versión de libre distribución para UNIX y Windows
- Sistemas distribuidos a gran escala (5000- 10000 nodos)
- Distingue entre nodos cliente y servidores dedicados
 - Los nodos cliente tienen que tener disco
- Ofrece a clientes dos espacios de nombres:
 - local y compartido (directorio /afs)
 - espacio local sólo para ficheros temporales o de arranque
- Servidores gestionan el espacio compartido
- Visión única en todos los clientes del espacio compartido

Sistemas Distribuidos
41

Fernando Pérez Costoya

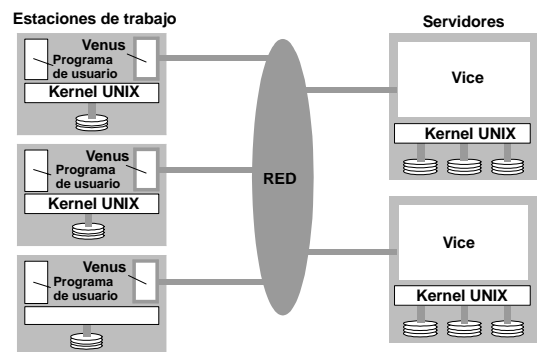
Estructura de AFS

- Dos componentes que ejecutan como procesos de usuario
- *Venus*:
 - ejecuta en los clientes
 - SO le redirecciona peticiones sobre ficheros compartidos
 - realiza las traducciones de nombres de fichero
 - resolución dirigida por el cliente
 - cliente lee directorios: requiere formato homogéneo en el sistema
- *Vice*:
 - ejecuta en los servidores
 - procesa solicitudes remotas de clientes
- Usan sistema de ficheros UNIX como almacén de bajo nivel

Sistemas Distribuidos
42

Fernando Pérez Costoya

Estructura de AFS



Sistemas Distribuidos
43

Fernando Pérez Costoya

Espacio de nombres compartido

- Los ficheros se agrupan en *volúmenes*
 - Unidad más pequeña que un sistema de ficheros UNIX
- Cada fichero tiene identificador único (UFID: 96 bits)
 - Número de volumen
 - Número de *vnodo* (dentro del volumen)
 - Número único: permite reutilizar números de vnodo
- Los UFID son transparentes de la posición
 - un volumen pueden cambiar de un servidor a otro.
- Soporte a la migración de volúmenes
- Estrategia de localización
 - número de volumen → servidor que lo gestiona
 - tabla replicada en cada servidor
 - cliente mantiene una cache de localización
 - si falla repite proceso de localización

Sistemas Distribuidos
44

Fernando Pérez Costoya

Acceso a ficheros

- Modelo de carga/descarga
 - En *open* servidor transfiere fichero completo al cliente
 - Versión actual: fragmentos de 64Kbytes
- *Venus* almacena el fichero en la cache local de los clientes
 - Se utiliza el disco local (la cache es no volátil)
- Lecturas/escrituras localmente sin intervenir *Venus*
 - Cache de UNIX opera aunque de manera transparente a AFS
- Cuando un proceso cierra un fichero (*close*)
 - Si se ha modificado se envía al servidor (*write-on-close*)
 - Se mantiene en cache local para futuras sesiones
- Modificaciones de directorios y atributos directamente al servidor

Coherencia de cache (1/2)

- Semántica de sesión
- Validación iniciada por servidor basada en *callbacks*
- Cuando cliente abre fichero del que no tiene copia local (o no es válida), contacta con el servidor
 - el servidor “anota” que el fichero tiene un *callback* para ese cliente
- Sigüientes aperturas del fichero no contactan con servidor
- Cuando cliente cierra un fichero que ha modificado:
 - Lo notifica y lo vuelca al servidor
 - Servidor avisa a los nodos con copia local para que la invaliden:
 - Evoca el *callback*
 - Solicitud en paralelo usando una multiRPC

Coherencia de cache (2/2)

- Cuando llega una revocación a un nodo:
 - procesos con fichero abierto continúan accediendo a copia anterior
 - nueva apertura cargará el nuevo contenido desde el servidor
- Los clientes de AFS asumen que los datos en su cache son válidos mientras no se notifique lo contrario
- El servidor almacena por cada fichero una lista de clientes que tienen copia del fichero en su cache:
 - la lista incluye a todos los clientes que tienen copia y no sólo a los que tienen abierto el fichero

Coda

- Descendiente de AFS orientado a proporcionar alta disponibilidad mediante replicación de volúmenes
- Lectura de cualquier copia – Escritura en todas
- Si red “partida”: cliente sólo actualiza copias accesibles
 - Se mantienen contadores de versión en cada copia de fichero
- En reconexión: se comparan contadores de las copias
 - Si no conflicto → se propaga a todas las copias la última versión
 - Si conflicto → reconciliación automática o manual
 - Ejemplo de automática: reconciliación de directorio
- Permite operación desconectada del cliente
 - Usuario puede sugerir qué ficheros deberían estar en cache
 - En reconexión: conciliación entre cache del cliente y servidores
 - Aunque no concebido para ello, es aplicable a computación móvil