Lezione 11

Cenni ai sistemi operativi distribuiti
4. File system distribuiti

- Caratteristiche di un file system distribuito
- Problematiche di progettazione
- Il Network File System
- Andrew File system

Caratteristiche di un file system

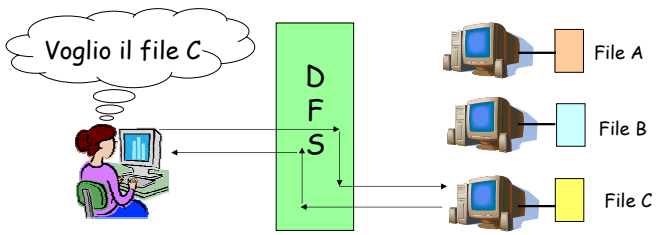
- Dati persistenti (i file)
- Spazio dei nomi gerarchico (o piatto) visibile a tutti i processi
- API per accedere e modificare i dati (open, close, read,…)
- Condivisione tra i processi attraverso meccanismi di controllo degli accessi (es. rwxrwxrwx di Unix)
- Accesso concorrente:
  - Sicuramente in lettura
  - Protocollo lettori/scrittori in aggiornamento
- Possibilita' di montaggio di altri file system

E' una implementazione di una memoria permanente
che consente la condivisone dei dati tra gli utenti

Un file system distribuito (DFS)

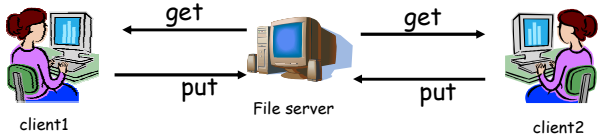
- Un file system distribuito e' una realizzazione distribuita del classico modello di file system

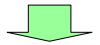
Estensione dello stesso tipo di condivisione al caso in cui i file sono distribuiti tra i siti di un sistema distribuito



Un primo esempio: FTP

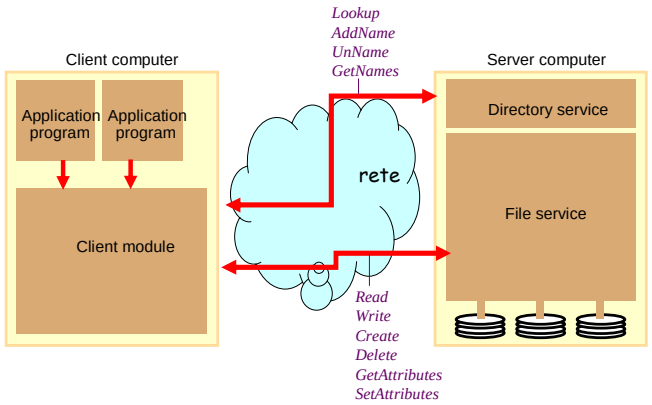
- L'utente ha a disposizione un insieme di operazioni sui file remoti che e' un sottoinseme delle operazioni sui file locali
- L'utente deve essere a conoscenza della locazione dei file
- Operazioni esplicite di upload e download (put e get)
- Coerenza di copie locali e remote a carico dell'utente
- Condivisione tramite "ftp anonymous"





Un po' poco per un DFS

Architettura generale di un DFS

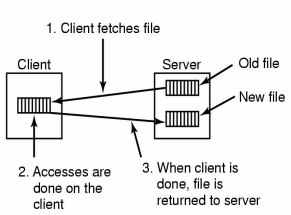


Problematiche di progettazione

- Nella progettazione di un file system distribuito bisogna affrontare molte problematiche:
  - Localizzazione dei file (quando richiesti, i file si devono spostare o rimanere sul server?)
  - Visione del file system (unica per tutti i client oppure no?)
  - Trasparenza (cosa deve sapere l'utente?)
  - Caching (come gestire la cache?)
  - Semantica (se 2 utenti modificano un file, quando devono essere visibili le modifiche?)
  - Fault tolerance (se un server diventa inaccessibile che succede?)

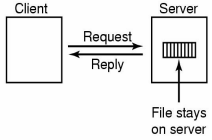
Un primo problema di progettazione

Modello upload/download



Il file viene scaricato per intero sul client, modificato e poi ricopiato sul server

Modello ad accesso remoto



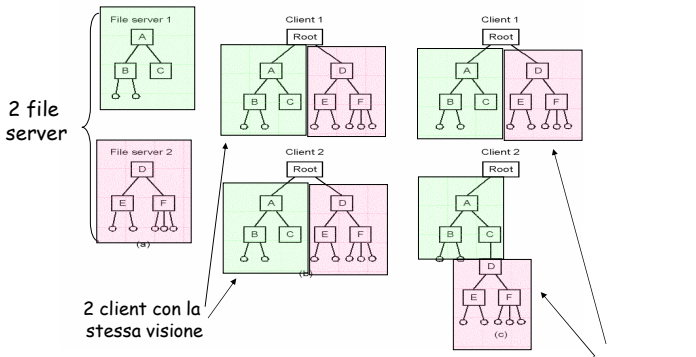
Il file rimane sul server e il client invia comandi perche' siano eseguiti in remoto mediante RPC

confronto

- Modello upload/download
  - Vantaggi:
    - semplicita' (facile implementare get e put)
    - efficienza (l'accesso avviene localmente)
  - Svantaggi:
    - possibile incoerenza (accesso concorrente al file)
    - spreco di risorse (se serve solo un pezzo del file)
- Modello ad accesso remoto
  - Vantaggi:
    - Assenza di incoerenza (1 sola copia del file)
    - Poca memoria richiesta (non ci sono duplicati)
  - Svantaggi:
    - inefficienza (sincronizzazione degli accessi e traffico di rete)

Un secondo problema

Visione uniforme delle directory vs visione non uniforme



confronto

- Stessa visione
  - ambiente familiare
  - Meno flessibile
  - Necessita' di una directory radice
- Visione differente
  - Piu' facile e flessibile
  - Ambiente che puo' "disorientare" gli utenti
  - Scelta piu' comune nei DFS

Deve esistere una directory radice?

Un terzo problema

- Trasparenza
  - Di accesso
    - (la modalita' di accesso deve essere indipendente dalla posizione fisica)
  - Di posizione
    - (la posizione fisica del file non deve essere nota all'utente)
  - Di replicazione
    - (l'utente non deve sapere quante copie del file esistono)
  - Di migrazione
    - (l'utente non deve sapere se il file viene spostato da un server ad un altro)

Non tutti sono facili da realizzare
(es. trasparenza di migrazione)

Schemi di nominazione

- I nomi dei file sono scelti come combinazione del nome dell'host e nome locale del file
  - Es: server1:/dir/nomefile oppure /server1/dir/nomefile
  - Garanzia di unico nome in tutto il file system
- Implementazioni:
  - Mount di directory remote (es. NFS)
  - Integrazione delle varie directory locali (es. AFS)

Un quarto problema: il caching

- Quando un utente richiede un file, il server su cui si trova e' identificato dallo schema di nominazione
- I dati sono trasferiti, ad esempio, mediante RPC
- Per migliorare le prestazioni e' necessario ridurre il traffico di rete generato da questi trasferimenti

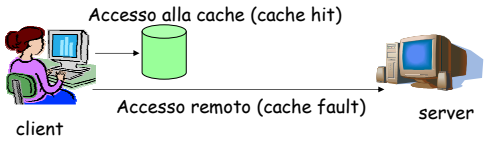
Caching dei file

- Se i dati non sono in cache: vengono trasferiti dal server
- Se sono in cache: gli accessi avvengono sulla copia in cache

Osservazione

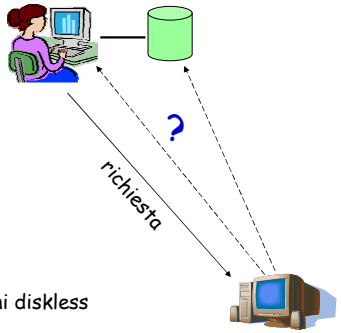
- La gestione di una cache per un file system distribuito e' molto simile ad una

Memoria virtuale di rete



Locazione della cache:

- Disco o memoria centrale?
- Cache su disco:
  - Piu' affidabile
  - Meno veloce
  - I dati non vengono persi
- Cache in memoria centrale
  - Implementabile su stazioni diskless
  - Piu' efficiente



Politica di aggiornamento della cache

- Scrittura diretta
  - Appena il client modifica il file nella cache, viene subito aggiornata anche la copia sul server
    - Affidabile ma con scarse prestazioni
- Scrittura ritardata
  - Le modifiche alla copia sul server avvengono ad intervalli di tempo
    - Poco affidabile
- Scrittura su chiusura
  - Si scrivono i dati sul server quando il file viene chiuso
    - Vantaggiosa per file aperti per lunghi periodi e modificati spesso

Coerenza della cache

- Il problema principale nell'uso della cache e' che, per sua natura, i file sono condivisi tra molti utenti

Possibilita' di incoerenza delle cache
(analogo al problema delle corse critiche)

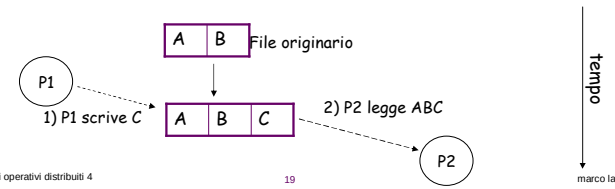
- Metodo iniziato dal client
  - Il client periodicamente controlla se la sua copia e' coerente con quella del server
- Metodo iniziato dal server
  - Il server tiene traccia dei file aperti con le rispettive modalita' e interviene se riscontra che uno stesso file e' aperto 2 volte in modalita' conflittuali (uso del protocollo lettore/scrittore)

confronto

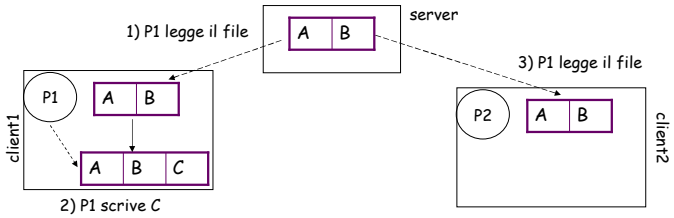
- Metodo iniziato dal client
  - Con che frequenza il client verifica la coerenza?
  - Es. ad ogni accesso, ad intervalli regolari, all'apertura del file
- Metodo iniziato dal server
  - Come reagisce il server?
    - Impone ai client di invalidare le copie in cache
    - Impone ai client di eliminare i file nelle cache dopo il loro uso
    - Ritarda l'apertura dei file
- In entrambi i casi le scelte determinano la semantica e l'efficienza

Un quinto problema:

- La semantica della condivisione
- Quando due o piu' utenti accedono allo stesso file e' necessario definire la semantica (cioe' definire quale comportamento e' errato e quale e' corretto)
- Esempio: consistenza sequenziale (semantica UNIX)
  - quando una read segue una write, la read deve restituire i dati appena scritti
  - Facile da realizzare nei monoprocessori



Cosa puo' accadere in un sistema distribuito



- Molto difficile e dispendioso realizzare una consistenza sequenziale in un ambiente distribuito
- Rilassare la semantica
- Accontentarsi di una semantica di sessione :
  - I cambiamenti sono visibili subito al processo che li ha fatti
  - Sono visibili agli altri processi quando ilfile viene chiuso

Un modello alternativo

- Se la semantica di sessione appare "superficiale" e puo' dare luogo ad inconsistenze, si puo':
  - Impiegare il modello download/upload
  - Il server pone un lock ad un file inviato ad un client
  - Un successivo client aspetta il rilascio del lock

Un sesto problema

- Se un server diventa inaccessibile che succede?
- Una soluzione e' la replicazione dei file (piu' copie disponibili in vari file server)
  - I guasti sono indipendenti ➔ miglioramento della fault tolerance
- Vincoli
  - Aggiornamento: una modifica deve propagarsi su tutte le copie
  - Trasparenza : l'esistenza di repliche deve essere invisibile all'utente (la differenza e' solo nello schema di nominazione)

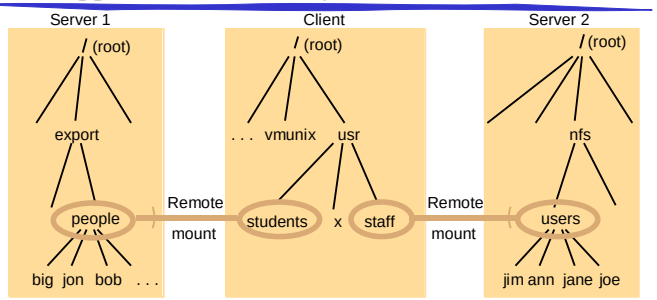
Esempi di File system distribuiti

- I piu' noti esempi di File System Distribuiti sono:
- Network File System (NFS)
  - E' il frutto di un progetto del 1984 della SUN Microsystems che aveva l'obiettivo di permettere l'accesso ai dischi remoti come se fossero dischi locali.
- Andrew File System (AFS)
  - E' un file system distribuito sviluppato nel 1983 alla Carnegie Mellon University come parte di un progetto per un sistema distribuito. Prende il nome da Andrew Carnegie e Andrew Mellon fondatori dell'universita'
- Nonostante la loro larga diffusione hanno caratteristiche molto diverse tra loro

Network File System (NFS)

- NFS permette a calcolatori con sistema Unix (ma non solo), che compongono un sistema distribuito, di condividere file, directory od un intero file system utilizzando il modello ad accesso remoto, come se si fosse in presenza di un unico file system logico
- Ogni server esporta una o piu' directory per far accedere i client da remoto
- I client accedono alle directory montandole come parte del proprio file system
- E' possibile far integrare tra loro f.s. eterogenei, sfruttando un protocollo per la rappresentazione dei dati (external data representation – XDR)

Montaggio delle directory



- Operazione di montaggio (mount):
  mount(remotehost, remotedirectory, localdirectory)
- Il server mantiene una tabella dei client che hanno montato il file system
- Il client mantiene una tabella dei file system montati che contiene:
  < IP address, port number, file handle>

NFS

- La versione 2 prevedeva che i server non dovessero conservare memoria degli accessi degli utenti (servizio senza stato). Eventuali meccanismi di blocco delle risorse (Lock) dovevano essere implementati esternamente al protocollo.
- La versione 3 introdusse il supporto per il trasporto delle informazioni per l'utilizzo di NFS su una WAN sebbene in maniera poco semplice ed efficiente.
- La versione 4 venne influenzata dall'AFS e includeva miglioramenti nelle prestazioni, aggiungeva un supporto migliorato alla sicurezza e introduceva un protocollo che teneva conto dello stato dei client
- NFS definisce due protocolli
  - Per il montaggio delle directory
  - Per l'accesso ai file e le directory

Protocollo per il montaggio

- Un client, per montare un directory, invia una rischiesta al server, specificando pathname e permessi per montare la directory
- Se il pathname e' valido e la directory e' esportata, il server restituisce al client un file handle con le informazioni sul file system, il disco, gli inode della directory e la sicurezza
- Il server mantiene una lista delle directory correntemente esportate e montate dai client (/etc/exports)
- Alcune versioni di Unix supportano l'Automount, che permette di montare automaticamente al boot le directory
- Automount gestisce la replicazione dei file mediante server alternativi ridondanti

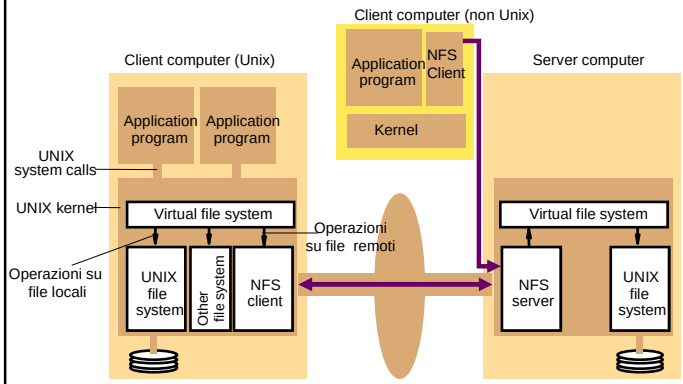
Protocollo per l'accesso a file e directory

- Il protocollo offre un insieme di RPC che permettono
  - Ricerca di un file in una directory
  - Lettura di un insieme di elementi di una directory
  - Lettura e scrittura di un file
  - Accesso agli attributi di un file
- Le versioni 2 e 3 non mantengono informazioni di stato ed ogni richiesta contiene tutte le informazioni per completare l'operazione
- Non sono necessarie le istruzioni di open e close

Implementazione di NFS

- NFS deve far parte del kernel?
- Non necessariamente: (ad es. in Windows 3.0, Win 95, primi MacOS, PocketPC era implementato come libreria utente)
- Per Unix e' pero' vantaggioso implementare NFS nel kernel:
  - Stesse chiamate di sistema per l'accesso locale e remoto (non c'e' bisogno di ricompilare l'applicazione)
  - Maggiore efficienza nella gestione della cache
  - Maggiore efficienza nell'accesso agli i-node

Architettura di NFS



Ottimizzazione – cache del client

- Il modulo client di NFS conserva in una cache i risultati delle operazioni di accesso ai file remoti
- La sincronizzazione non e' garantita nel caso che due client accedono al file
- Controllo di validita' della cache basata su marche temporali

$(T - Tc < t)$ or $(Tm_{client} = Tm_{server})$

Le marche temp. (e quindi le copie) sono uguali

Il dato e' recente

t	periodo di aggiornamento (prefissato 3-30 sec)
Tc	marca temporale dei dati nella cache del client
Tm	marca temporale della copia del file sul server
T	tempo corrente

Riduce l'inconsistenza ma non la elimina

NFS: conclusioni

- Eccellente esempio di semplice, robusto ed efficiente file system distribuito
  - **Trasparenza di Accesso**: eccellente. Le API di Unix sono le stesse per le chiamate locali e remote
  - **Trasparenza di Locazione**: non garantita ma possibile con una appropriato montaggio delle directory remote
  - **Concorrenza** : adeguata per la maggior parte dei casi. Quando i file sono modificati da piu' client ci puo' essere inconsistenza.
  - **Replicazione dei file**: limitata ai soli file in lettura
  - **Fault tolerance**: buona. E' sospeso il servizio solo del server che diviene non disponibile e il recovery e' aiutato dall'assenza di stato (vers. 2 e 3)
  - **Efficienza**: buona. La maggior parte del tempo di accesso e' speso nella risoluzione dei nomi (utile la cache nel client). Letture e scritture sui file pesano per al piu' il 5% del tempo.
  - **Scalabilita'**: buona. File system molto grandi possono essere suddivisi su piu' server.

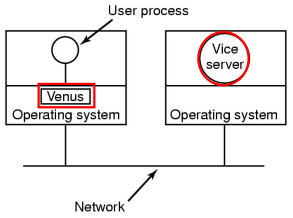
Andrew File System - AFS

- AFS e' un File System distribuito che permette la condivisione di file tra macchine UNIX su reti LAN e WAN utilizzando il modello upload/download
- Obiettivo del progetto: fornire a studenti e docenti della Carneige Mellon Univ. un ambiente Unix-like con un unico file system condiviso
- AFS e' scalabile. E' possibile connettere migliaia di workstation
- Solitamente si tengono distinte le workstation dai server per migliorare le prestazioni.
- AFS supporta varie versioni di Unix (AIX, HP-UX, Ultrix, SUN-OS,...)

Organizzazione

- AFS si integra con il sistema operativo a livello di kernel; esso modifica la struttura interna del file system di Unix.

necessaria una particolare installazione che comporta la ricostruzione del kernel della macchina



- Nel kernel del client c'e' una porzione aggiunta di codice chiamata Venus
- I file server, detti Vice, girano in spazio utente

AFS: architettura

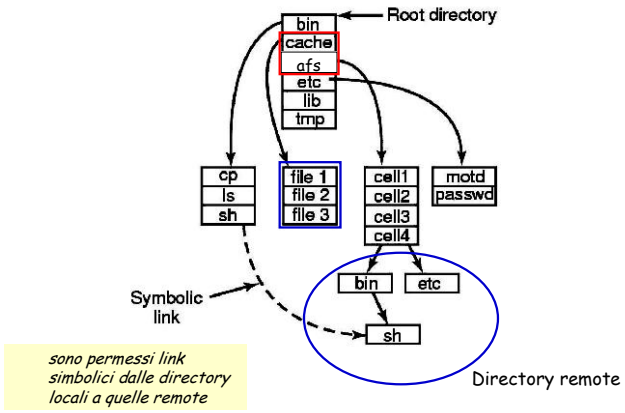
- I client sono presentati con uno spazio dei nomi partizionato: uno spazio dei nomi locali e uno spazio dei nomi globali
- Su ogno client,lo spazio dei nomi visibili ai programmi utente appare come un albero UNIX tradizionale, con l'aggiunta delle directory /afs e /cache
  - La directory /cache contiene i file remoti posti nella cache,
  - La directory /afs contiene i nomi delle workstation remote, raggruppate in cellule, sotto cui si trovano i rispettivi file system
  - I file system remoti sono montati in /afs; le altre directory e file sono strettamente locali e non condivisi

Le cellule

- Introdotte in AFS 3
- Cellula = gruppo di ws geograficamente vicine (ad es. ws di uno stesso dipartimento o universita') e amministrate in comune
- Piu' cellule possono coprire un territorio molto vasto

scalabilita' di AFS a migliaia di ws
anche molto distanti geograficamente

File system di AFS



sono permessi link simbolici dalle directory locali a quelle remote

AFS: idea di base

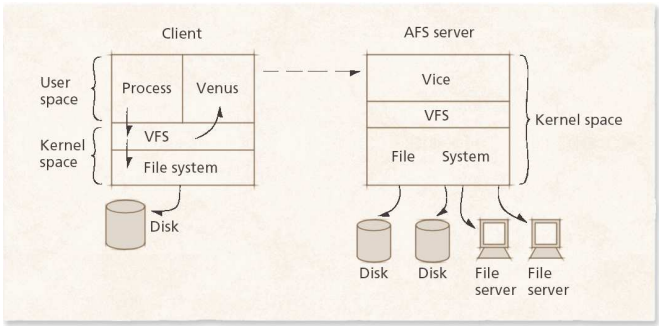
- Assunzione 1 : i file sono usati molto in lettura e poco in scrittura
- Assunzione 2: la maggior parte dei file e' piccola

IDEA:

Fare in modo che ogni utente lavori il piu' possibile localmente e interagisca il meno possibile con il resto del sistema

- All'apertura di un file, Venus cattura la open e trasferisce tutto il file sul disco locale
- Il file descriptor si riferisce al file locale
- Tutte le operazioni di read e write sono sul file locale
- Alla chiusura del file Venus trasferisce l'intero file sul server

Organizzazione di AFS



Semantica

- Venus scarica un file nella directory /cache locale
- read e write locali
- Alla chiusura il file viene trasferito sul server
- Se nel frattempo un altro processo accede al file le modifiche non sono visibili

AFS usa una semantica di sessione

Le modifiche sono visibili agli altri processi solo dopo la chiusura del file

Riepilogo delle operazioni sui file in AFS

User process	UNIX kernel	Venus	Net	Vice
open(FileName, mode)	If *FileName* refers to a file in shared file space, pass the request to Venus.	Check list of files in local cache. If not present or there is no valid *callback promise*, send a request for the file to the Vice server that is custodian of the volume containing the file.	→	Transfer a copy of the file and a *callback promise* to the workstation. Log the callback promise.
	Open the local file and return the file descriptor to the application.	Place the copy of the file in the local file system, enter its local name in the local cache list and return the local name to UNIX.	←	
read(FileDescriptor, Buffer, length)	Perform a normal UNIX read operation on the local copy.			
write(FileDescriptor, Buffer, length)	Perform a normal UNIX write operation on the local copy.			
close(FileDescriptor)	Close the local copy and notify Venus that the file has been closed.	If the local copy has been changed, send a copy to the Vice server that is the custodian of the file.	→	Replace the file contents and send a *callback* to all other clients holding *callback promises* on the file.

Il caching

- Il Cache Manager e' un processo che gira sulle macchine client e gestisce la cache locale
- Dopo la chiusura, il file viene trasferito sul server, ma rimane anche nella cache locale in modo da evitare successivi download in caso di altri utilizzi

Che succede se il file nel frattempo
e' stato modificato da un altro processo?

- Due approcci
  - Verifica della coerenza iniziata dal client (AFS 1)
  - Verifica della coerenza iniziata dal server (AFS 2 e 3)

Verifica iniziata dal client (AFS1)

- Ogni volta che Venus trova e apre un file nella cache locale contatta il server per assicurarsi della validita' del file

Vengono generate molte richieste al server
anche quando il file non e' stato modificato

Verifica iniziata dal server (AFS2 / AFS3)

- Il server tiene traccia dei file che i client hanno nelle proprie cache
- Ogni volta che un file viene chiuso (e scaricato sul server), il server comunica ai client di invalidare la copia del file in loro possesso
- In caso di nuovo utilizzo i client scaricano la nuova copia nella cache

Minore comunicazione tra client e server

Sicurezza

- AFS garantisce la sicurezza utilizzando due modelli
  - Password: per utilizzare i file nella directory /afs e' necessaria una password. In caso di successo il Cache Manager riceve dal server un identificativo utilizzato per le successive transazioni
  - Lista di controllo: Ad ogni directory di /afs e' associata una tabella che ne specifica i diritti di accesso. Cio' permette agli utenti di restringere l'accesso ai propri file che si intende condividere.

AFS: conclusioni

- Trasparenza di locazione e di accesso: ogni client ha la stessa visione dei file poiche' AFS ha una radice comune
- Prestazioni: la cache locale riduce il traffico di rete
- Scalabile: mantiene le stesse prestazioni sia su piccoli cluster sia su grandi installazioni
- Sicurezza: piu' sicuro di NFS
- Concorrenza: garantita secondo la semantica di sessione
- Fault tolerance: se il server diventa indisponibile non e' possibile aggiornare il file ma e' possibile continuare ad usare la copia nella cache locale