

Sistemi Operativi II

Corso di Laurea in Ingegneria Informatica

Facolta' di Ingegneria, Universita' "La Sapienza"

Docente: Francesco Quaglia

File system distribuiti:

1. Nozioni di base
2. Aggiornamento dati
3. Gestione delle cache
4. Esempi (NFS, Sprite, AFS)

Introduzione

- i file system distribuiti sono sistemi di archiviazione la cui memoria di massa risiede su dispositivi di macchine distinte
- successo dei file system distribuiti nasce dall'alto costo della memoria di massa intorno alla metà degli anni ottanta
- quindi nei sistemi aziendali si preferiva condividere questa risorsa tra gli utenti in modo da limitarne l'acquisto in grandi quantità (stazioni di lavoro diskless)
- per esempio il software applicativo doveva risiedere solo in una macchina (server), l'utente interessato all'esecuzione di quel programma, di fatto scaricava prima il programma dal server sulla sua macchina locale e poi lo mandava in esecuzione
- vantaggi attuali primariamente di ridotto costo di amministrazione delle applicazioni

Proprietà auspicabili

Trasparenza di locazione

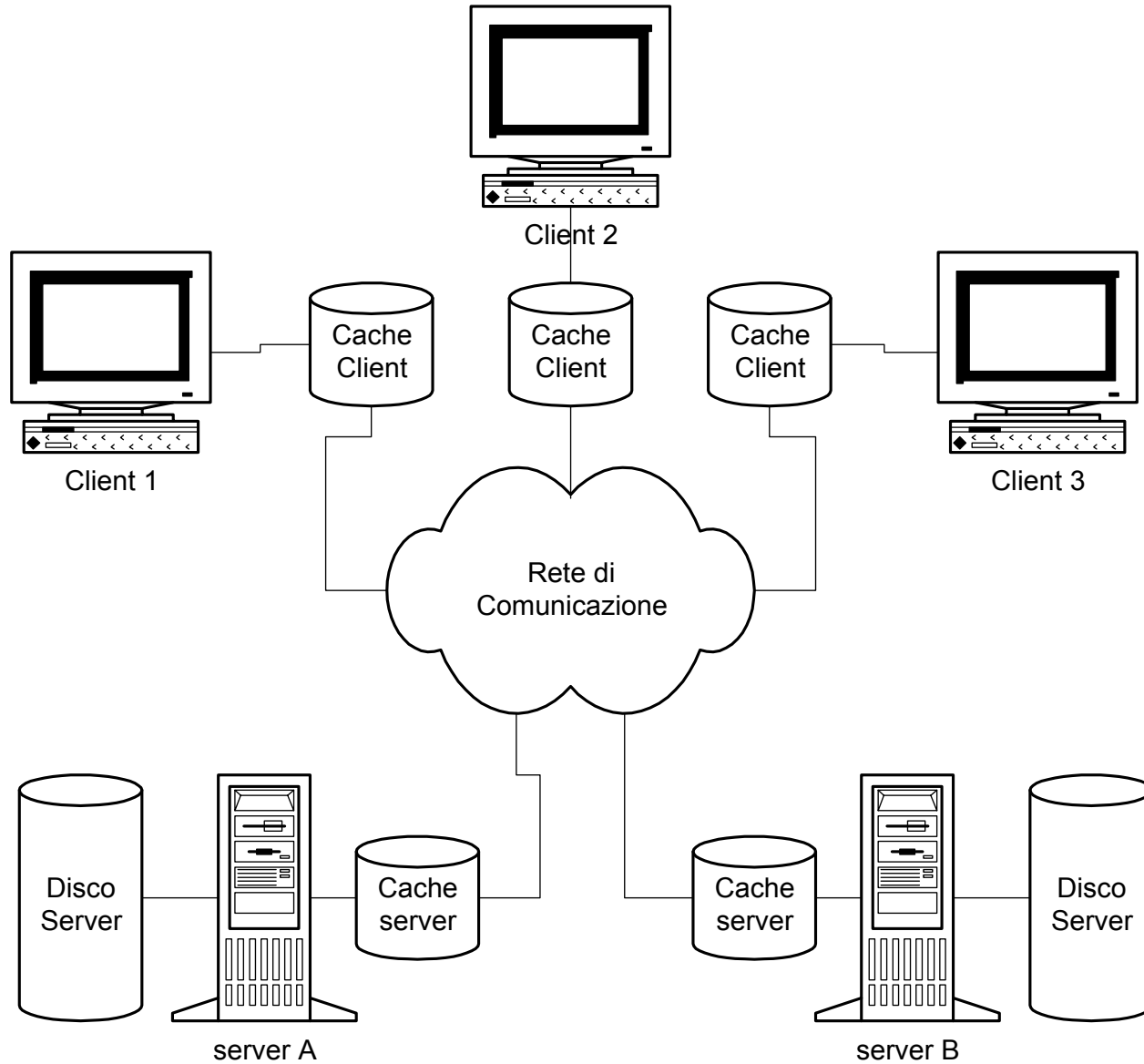
- l'utente deve dare il comando di esecuzione del programma come se questo fosse un programma locale, senza rendersi conto cioè delle operazioni descritte precedentemente
- al più può notare una latenza nell'avvio del programma
- in altre parole un utente deve poter eseguire operazioni sui file di un file system distribuito come se questi fossero dei file locali

Indipendenza di locazione

- il nome di un file non deve essere modificato se cambia la locazione fisica del file stesso

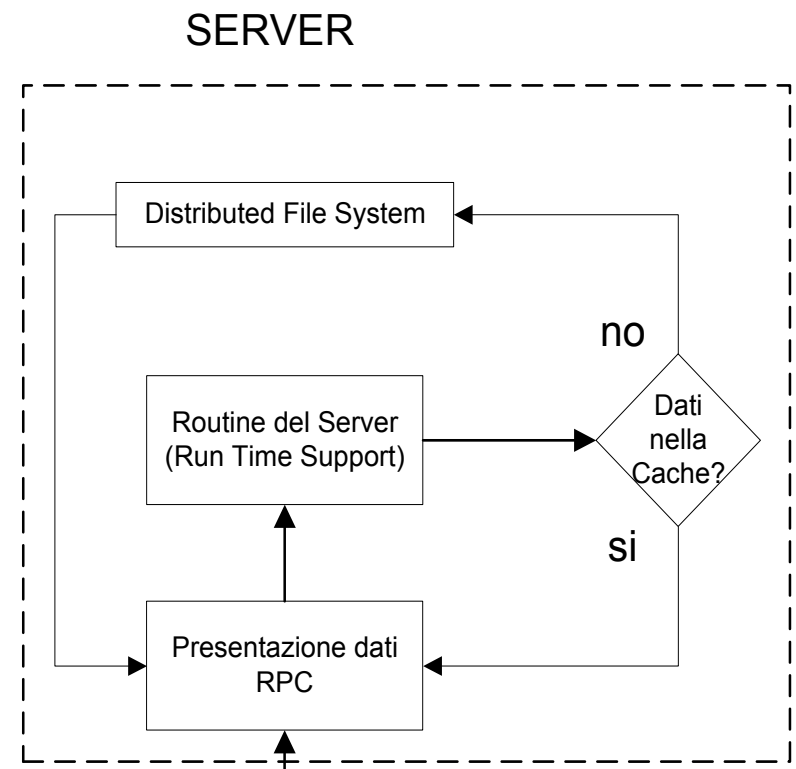
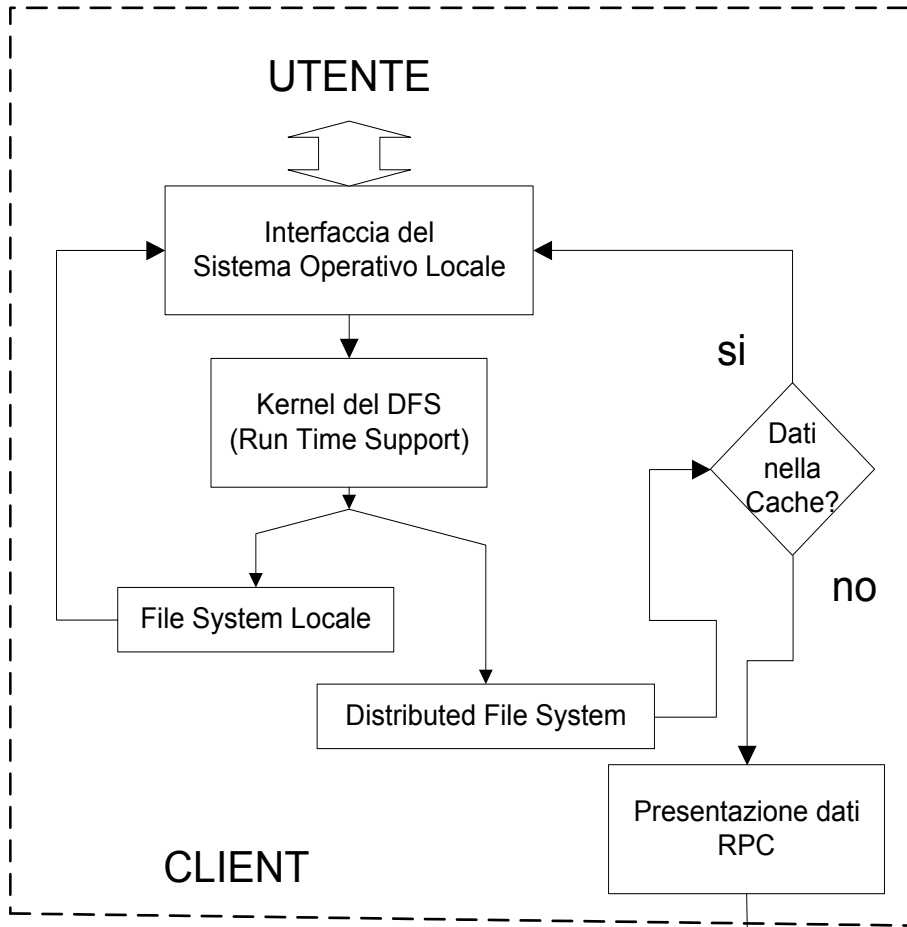
Gestione della replicazione dei file

Architettura



Funzionamento

- ogni client di un file system distribuito possiede un kernel (o supporto a run time) che intercetta tutte le richieste per accesso a file (apertura, chiusura, scrittura e lettura)
- il kernel riconosce se la richiesta riguarda un file del file system locale o di un file del file system distribuito
- nel secondo caso controlla se il dato è nella memoria cache locale al client, in caso affermativo invia i dati richiesti verso l'utente
- in caso contrario, il kernel invoca una RPC all'interno del server per ottenere i dati
- una volta recuperati i dati il server prepara il pacchetto da inviare al client come risultato della RPC



Aggiornamento della copia primaria

Write through

- ogni volta che un client scrive dati su di un file, la scrittura viene notificata al server che gestisce la copia primaria del file
- approssimazione della semantica UNIX

Delayed write

- le scritture di dati da parte di un client vengono notificate al server che gestisce la copia primaria solo periodicamente
- approssimazione (più lasca) della semantica UNIX

Write on close

- le scritture di dati da parte di un client vengono notificate al server che gestisce la copia primaria solo al termine della sessione di lavoro del client su quel dato file
- semantica di sessione

Coerenza della cache

Approccio “client initiated”

- il client verifica se i dati nella cache sono coerenti con la copia primaria, in caso negativo i dati vengono nuovamente prelevati
- il controllo può essere effettuato ad ogni accesso (approssimazione semantica UNIX), all’inizio della sessione di lavoro sul file oppure su base periodica

Approccio “server initiated”

- il server notifica ai client che il contenuto della loro cache è non più coerente per effetto di scritture da parte di altri client (**callback**)
- la notifica è ad ogni aggiornamento, periodica o sulla chiusura della sessione di lavoro
- il server potrebbe decidere di invalidare il caching al lato client nel caso in cui una sessione di lavoro venga attivata per in certo file

Replicazione dei file

Obiettivi

- aumento della disponibilità dei dati
- miglioramento delle prestazioni (bilanciamento del carico su più server)

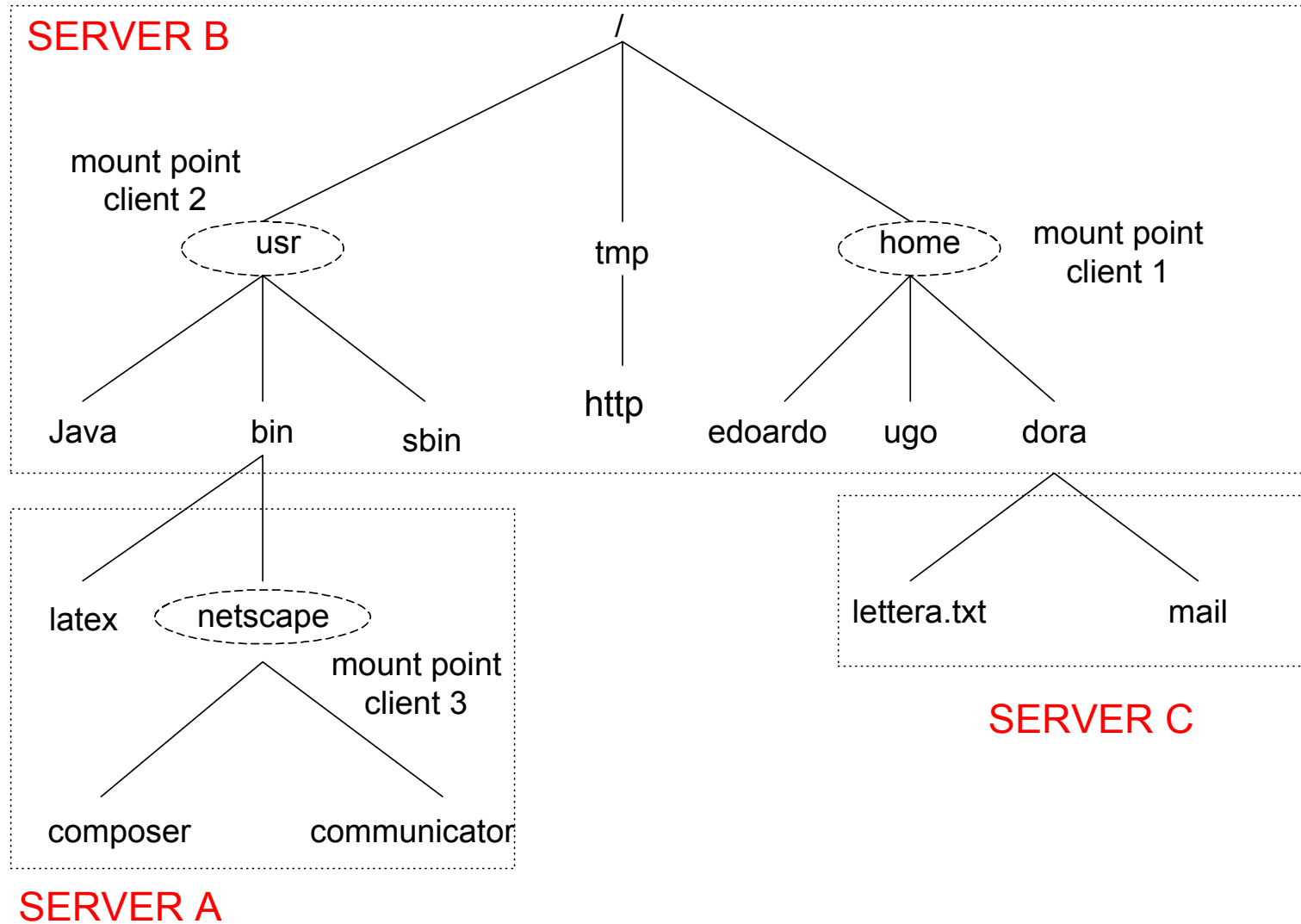
Consistenza delle repliche

- aggiornamento sequenziale: effettuato da parte del client su tutte le copie (problemi in caso di caduta del client)
- replicazione di copia primaria: gli aggiornamenti vanno solo sulla copia primaria, il server che la gestisce si fa carico di aggiornare le altre copie (tipico di applicazioni read intensive)
- quorum: aggiornamenti e letture avvengono su $N/2+1$ copie (dove N è il numero totale) per evitare problemi di caduta del server primario

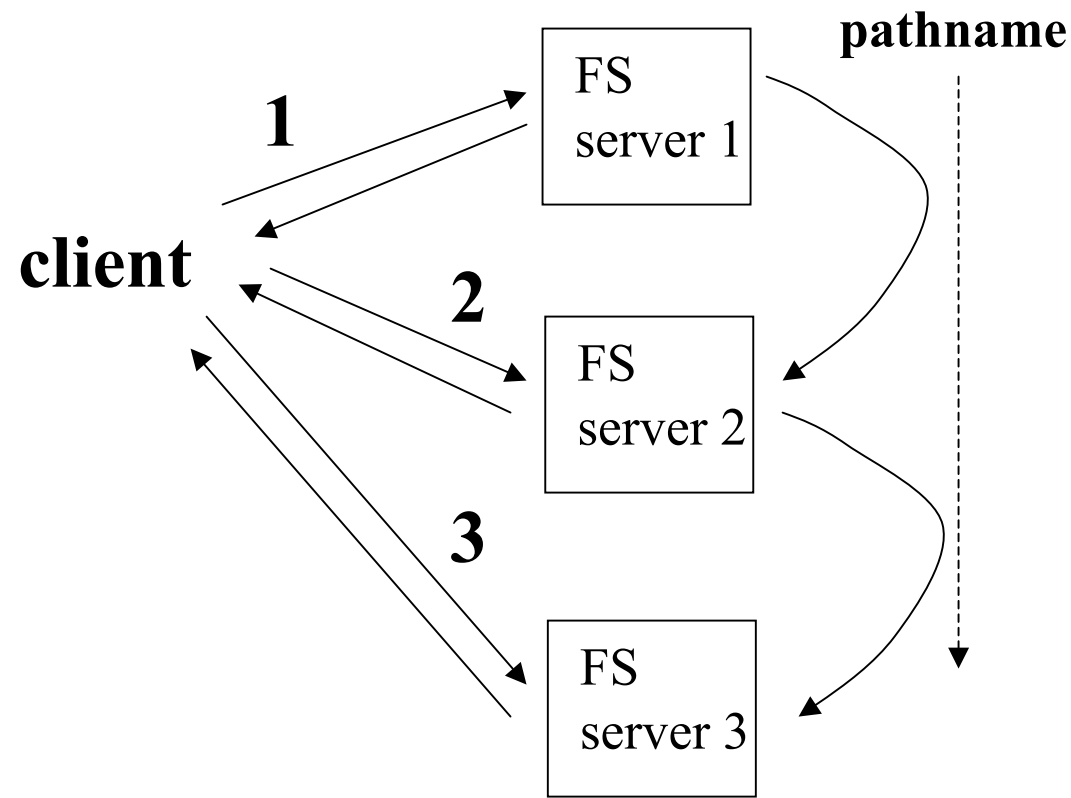
Montaggio

- ogni server ha un proprio spazio di nomi di file ed una propria gerarchia
- questi spazi devono essere collegati per formare un'unica struttura gerarchica vista dai client
- i client si collegano a questa struttura in punti particolari detti **mount point** (mantenuti in una **mount table** gestita dal kernel del DFS)
- i mount point corrispondono a delle directory nella struttura gerarchica unica e permettono ad un client di vedere tutti file al di sotto del suo, o dei suoi, mount point
- per fissare un mount point, il client esegue l'operazione di collegamento alla directory di interesse (**mounting**)
- questa operazione avrà successo solo se il server darà opportuni diritti al client per eseguire quel mounting

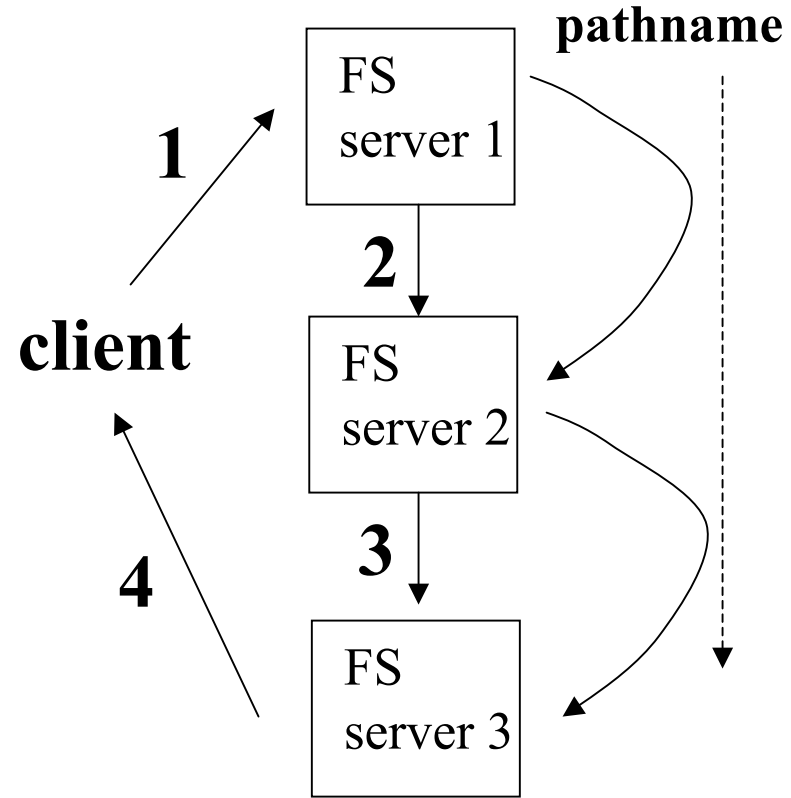
Un esempio



Ricerca dei dati su montaggio a cascata



**ricerca iterativa
(RPC compliant)**



**ricerca automatica
(non RPC compliant)**

Trasferimento di dati

- se ci si limita a trasferire i pochi dati per ogni richiesta, il tempo che viene preso dal server per trovare i dati da trasferire sul disco (seek time) e dal protocollo per trasferire i dati stessi (segmentazione, riassemblamento, instradamento ecc.) non viene ammortizzato
- quindi si preferisce inviare un blocco di dati contigui, contenente i dati richiesti, appartenenti allo stesso file
- questo permette di ottimizzare il seek time e l'esecuzione del protocollo di scambio dati
- il trasferimento di dati contigui ha aumentato le prestazioni dei file system distribuiti poiché segue un principio semplice ed efficace: una volta che un file è stato aperto, questo viene, molto spesso, visitato nella sua interezza da un utente

Network file system (NFS)

Caratteristiche di base

- NFS è l'applicazione che ha portato al successo SUN Microsystem alla metà degli anni ottanta
- NFS nasce su cluster di workstation UNIX, ma ormai, a livello client, viene supportato da tutti i più comuni sistemi operativi (Windows, Macintosh, LINUX)
- montaggio non trasparente, accesso trasparente
- montaggio a cascata non transitivo
- FS esportabili definiti in **/etc/exports**, insieme ad informazioni su chi può importare
- **/etc/fstab** specifica cosa montare
- basato su UDP a livello dell'architettura RPC
- accesso alla home directory da qualsiasi macchina nell'architettura

NFS

Modalità di funzionamento

- un utente che desidera accedere ad un file manda la richiesta ad un file system virtuale presente sull'elaboratore (gestione di v-nodes tramite informazioni in /etc/mstab)
- questo controlla se il file desiderato è presente nella macchina locale oppure è un file di NFS
- nel secondo caso, la richiesta viene inviata, attraverso il virtual file system, ad un NFS client che la rilancerà all' NFS server attraverso una RPC
- il server NFS controlla i diritti di accesso al file da parte dell'utente e serve la richiesta inviando una copia del file al client NFS che immagazzinerà tale copia nella cache locale
- read/write atomico su al più 1500 bytes (dovuto al protocollo ethernet)

NFS: gestione dei dati

- delayed write per la notifica di aggiornamenti verso il server
- il server NFS non tiene traccia delle copie inviate ai vari client
- uso di timestamp per invalidare copie obsolete dei dati (in ogni caso l'invalidazione avviene sempre per effetto di azioni del client, ad esempio scritte su un certo file)
- il timestamp di un file viene incrementato dal server di uno ogni volta che viene eseguita una modifica e viene attaccato ai dati quando il server li spedisce ad un client
- quando le modifiche di un file arrivano al server esse hanno attaccato il timestamp associato originariamente dal server alla copia
- se il timestamp associato alle modifiche è minore del timestamp del file nel server, significa che qualcuno ha effettuato delle modifiche al file e che la copia nella cache del client deve essere invalidata

Sprite File System (SFS)

- simile ad NFS
- ogni 5 secondi il kernel di SFS controlla la cache del client alla ricerca di dati che non sono stati modificati negli ultimi 30 secondi, prende questi dati e li invia al server
- quindi dati modificati non saranno inviati al server prima di 30 secondi dalla loro modifica oppure prima di essere espulsi dalla cache del client per essere rimpiazzati con dati più recenti
- questa politica nasconde al server tutti quei dati le cui modifiche non hanno tempo di vita superiori ai 30 secondi riducendo notevolmente il traffico di rete
- il server invia al client un messaggio dicendogli che alcuni suoi dati sono obsoleti
- il client può decidere se eliminarli dalla cache o scaricarsi dal server una versione recente
- il problema delle modifiche concorrenti viene risolto non ammettendo (o ritardando) il caching dei file che sono aperti in scrittura

Andrew File System (AFS)

- ideato alla Carnegie Mellon University e successivamente commercializzato da Transarc, una sussidiaria di IBM
- la sua caratteristica più importante sta nel fatto che AFS è un “unico” file system su scala mondiale
- ci sono nel mondo circa un migliaio di server e decine di migliaia di client tutti uniti in un unico file system
- AFS è organizzato in celle, ogni cella rappresenta una directory di primo livello del file system
- una cella mette a disposizione di un client software applicativo ed eventuali file dati
- adotta caching di file in grosse porzioni
- si basa su una semantica di sessione con callback