File System Distribuiti (DFS)

Sommario

- Background
  - File service & directory service
- Semantica della condivisione
- Implementazione di DFS
- Caching
- Caso di studio: NFS

Background

- Interfaccia del file system verso l'utente
  - File service & Directory service
    - Servizi offerti dal/dai file server
- Modello di accesso a file (file service)
- Naming (directory service)

File service e directory service

- Interfaccia del file system verso l'utente
  - Per gli utenti è importante quale servizio viene offerto, non come viene implementato
  - In molte implementazioni due componenti separate
  - File service = funzioni legate a file individuali
    - Lettura, scrittura, …
    - Aspetto principale: modello di accesso a file
  - Directory service = funzioni legate a directory
    - Creazione, spostamento, cancellazione file e cartelle, …
    - Aspetto principale: naming

File service

- Modello di accesso a file
  - Upload/download
  - Accesso remoto

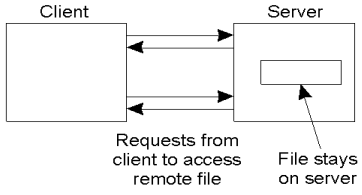
Modello di accesso a file

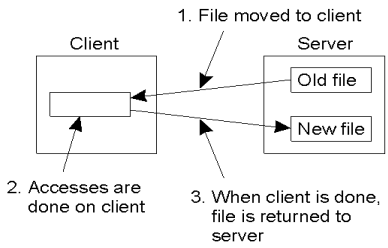
- Modello upload/download
  - Intero file viene trasferito dal file server e memorizzato localmente (memoria o disco)
  - Primitive
    - read_file() per scaricare il file in locale
    - write_file()per trasferire un file verso file server
  - Vantaggi
    - Semplice: no specifiche interfacce di trasferimento
  - Svantaggi
    - Inefficiente: richiede spazio locale per intero file
      - Magari basta solo una parte del file

Modello di accesso a file

- Modello ad accesso remoto
  - File system effettivamente su server
  - Primitive
    - tutti le possibili operazioni su file
  - Vantaggio
    - Memoria locale solo per porzioni di file
  - Svantaggio
    - Ogni operazione richiede comunicazione con server

Modello di accesso a file





Accesso remoto **Upload-download**

Directory service

- Naming
  - Directory service definisce sintassi per comporre i nomi di file e directory, per localizzarli nel file system, per crearli, cancellarli e spostarli
  - Tutte le macchine devono avere la stessa visione della gerarchia del file system?
    - Generalmente no

Naming

- Naming = mapping tra oggetti (file/dir) logici e fisici
- Proprietà essenziale: trasparenza
  - Trasparenza della locazione
    - Il nome non rivela la locazione (fisica) del file server
    - Non so se file remoto o locale
    - Es: /server1/a/b/c (server1 dov'è?)
  - Indipendenza della locazione
    - Il nome non deve essere modificato se cambia il server
    - Difficile da realizzare
    - Es: /server1/a/b/c dipende dalla locazione!

Schemi di naming

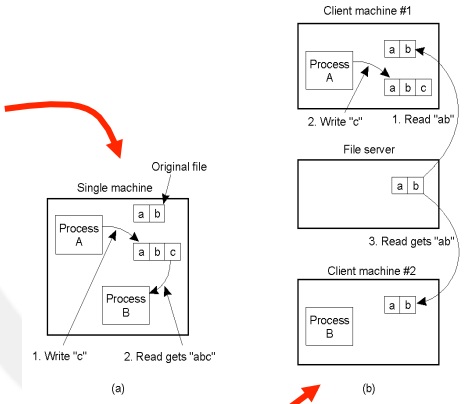
- Nomi = host/nome locale
  - Nome unico nel sistema
  - Non trasparente
- “Attach” di directory remote su quelle locali
  - Concetto di mount
  - Trasparente per directory pre-montate
  - Non indipendenza della allocazione
- Integrazione totale dei vari file system
  - Unica struttura globale di naming che copre tutti i file del file system
  - Difficile da realizzare

Semantica della condivisione

- Cosa succede se un processo apre un file in scrittura e un altro apre lo stesso file in lettura?
  - Semantica Unix
  - Semantica di sessione
  - File immutabili
  - Transazioni atomiche

Semantica della condivisione

- Univoca su sistemi tradizionali
  - Una lettura successiva ad una scrittura ritorna sempre il valore appena scritto
- In sistemi distribuiti (con caching) il valore ritornato può essere non aggiornato



Semantica della condivisione

- Semantica UNIX
  - Uguale a semantica uniprocessore
- In sistemi distribuiti ottenibile se
  - Un unico file server esegue sequenzialmente le richieste
  - No caching
  - (Minime inconsistenze per ritardi su rete)
- Scarsa efficienza → caching indispensabile!
  - Propagazione immediata delle modifiche al server non fattibile
  - ➡ rilassamento della semantica

Semantica della condivisione

- Semantica di sessione
  - Modifiche su file visibili “agli altri” soltanto dopo la chiusura
- Problemi in caso di modifica simultanea da parte di più processi
  - Risultato tipico
    - ultima chiusura → ultimo valore

Semantica della condivisione

- File immutabili
  - Non è possibile modificare un file (directory sì)
  - Uniche primitive: create() e read()
  - Scrittura = creazione di un file “modificato”
    - Scrittura e lettura agiscono su file diversi!
- Problema di rimpiazzamento simultaneo
  - Risultato tipico: ultima chiusura → ultimo valore
- Problema di rimpiazzamento durante lettura
  - Soluzioni
    - Alla UNIX → lettore continua utilizzo vecchio file
    - Accorgersi della scrittura e impedire letture successive alla modifica

Semantica della condivisione

- Transazioni atomiche
  - Accesso protetto da BEGIN_TRANSACTION/ END_TRANSACTION
    - Tutte le operazioni su file della transazione sono eseguite in ordine e senza interferenze di altre transazioni
  - Se due o più transazioni iniziano nello stesso istante, il sistema assicura che il risultato finale sia consistente (serializzazione delle transazioni, in ordine arbitrario)

Semantica della condivisione

Modello	Caratteristiche
Semantica UNIX	Ogni operazione su un file è immediatamente visibile a tutti i processi
Semantica di sessione	Nessuna modifica è visibile agli altri processi fino a che il file non viene chiuso
File immutabili	Modifiche non possibili; semplifica condivisione e replica
Transazione	Tutte le modifiche avvengono atomicamente

IMPLEMENTAZIONE DI DFS

Implementazione di DFS

- Aspetti coinvolti
  - Tipologia di accesso
    - Per capire quali scelte effettuare nell'implementazione
  - Server stateless vs. stateful
  - Caching
  - Replica di file
  - Protocolli di modifica

Tipologia di accesso

- Studio statistico [Satyanarayanan 81]
  - La maggior parte dei file < 10K
    - Ha senso trasferire file anziché blocchi
  - Lettura più comune di scrittura
  - Scritture/letture ad alta località
    - Accesso casuale raro
  - Tempo di vita (medio) di un file molto breve
    - Ha senso creare file localmente sui client (es.: compilatore)
  - Condivisione di file atipica
    - Ha senso fare caching su client
  - Esistono classi di file con diverse problematiche
    - File temporanei, file ordinari, mailbox, file binari, …

Struttura del file service

- Problemi implementativi da affrontare
  - C'è differenza tra client e server?
    - In NFS no
    - Altri approcci prevedono che server e client siano diversi sia dal punto di vista SW che HW
  - File e directory service separati?
    - Funzioni indipendenti, può avere senso separarli
    - Separazione richiede maggior numero di comunicazioni
  - File (o directory) server devono mantenere informazioni sullo stato dei client?
    - Stateless vs. stateful server

File Server stateful

- Mantiene informazioni sullo stato del client
- Schema (apertura di file)
  - Client apre un file
  - Server preleva informazioni sul file da disco, memorizza e fornisce al client un identificatore di connessione unico e il file stesso
  - Identificatore usato per accessi successivi fino al termine della sessione
  - Server rientra in possesso della memoria usata da client non attivi
- Alcuni servizi richiedono stateful
  - Es.: lock del file, cache nel server, …

File Server stateless

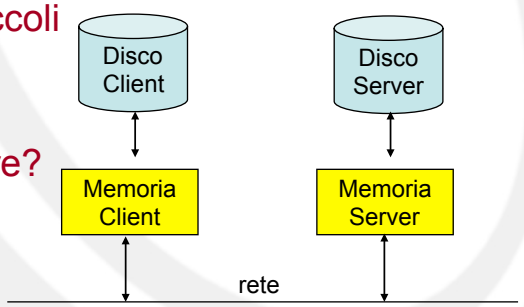
- Evita informazioni sullo stato del client
- Schema (apertura di file)
  - Client apre un file
  - Ogni richiesta identifica il file e relativa la posizione dell'accesso
  - Terminato l'accesso il server libera ogni informazione memorizzata
- Ogni accesso è autoconsistente
  - No concetto di "sessione"!

Differenze (stateful vs. stateless)

Vantaggi server stateless	Vantaggi server stateful
Tolleranza ai guasti (crash del server o dei client non problematico)	Messaggi più compatti (messaggi non contengono nomi di file)
No operazioni tipo open()/close() (riduce numero di messaggi)	Migliori prestazioni medie (informazioni su file in memoria, in Unix i-node)
Minimo spreco di spazio nelle tabelle del server	Possibilità di read-ahead (server sa se un file è aperto per accesso sequenziale)
Nessun limite sul numero di file aperti	Facile realizzare idempotenza (richieste duplicate intercettabili)
	Possibilità di file locking

Caching

- Caching irrinunciabile per motivi di prestazioni
  - Trasferimento di molti dati occasionalmente meglio di molti piccoli trasferimenti
- Problema
  - Dove memorizzare?
    - Cache del server
    - Cache dei client



Server caching

- Server mantiene copia dei file “recenti” nella sua memoria
- Soluzione più semplice
- Trasparente ai client
- Nessun problema di consistenza
  - C’è un unica copia dei file
- Inefficiente
  - Eliminiamo accessi al disco del server, ma…
  - … rimane un accesso di rete per ogni accesso a file!

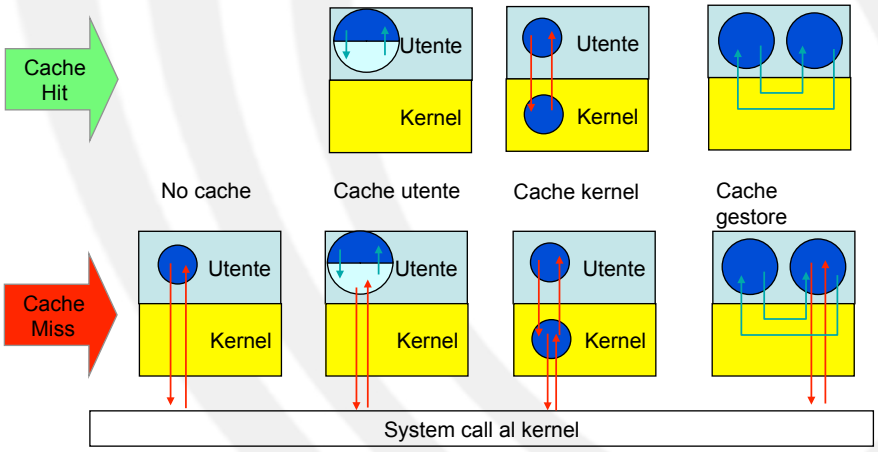
Client caching

- Client mantiene copia dei file “recenti” nella sua memoria/disco
- Soluzione più complessa
  - “Dove” realizzare la cache?
  - Problema della coerenza della cache

Client caching

- Dove memorizzare i dati?
  - Spazio di indirizzamento del processo
    - Sensato se il processo accede ad un file in modo ripetuto
    - Cache “muore” con il processo
  - Kernel
    - Chiamata a kernel per ogni accesso
    - Cache sopravvive al processo
  - Processo gestore di cache
    - Kernel non contiene codice per il file system
      - Facile da programmare e flessibile (microkernel)
    - Hit “virtuali” se posso fare swap del processo gestore

Dove mantenere i dati?



Consistenza della cache

- La copia nella cache locale è consistente (uguale) con la copia su server?
- Necessari schemi di aggiornamento della cache del server
  - Write-through
  - Delayed write
  - Write on close
  - Controllo centralizzato

Write through

- Appena un blocco della cache è modificato viene propagato immediatamente al server
- Sicuro, ma poco efficiente
  - Traffico ~ uguale al caso senza cache
- Potenziali problemi per cache locali “vecchie”
  - Processo su macchina A legge f e termina
  - Poi processo su macchina B modifica f e termina
  - Infine nuovo processo su macchina A legge f, ma trova la cache vecchia!!
  - Soluzione: controllo sulla data del file

Delayed write

- Aggiornamenti sul server scritti “in gruppo”, ad intervalli regolari
  - Pochi trasferimenti lunghi sono meglio di tanti corti
  - Evita aggiornamenti di file temporanei
- Affidabilità scarsa
  - Dati non scritti vanno persi su crash di client
- Semantica ambigua
  - Quando un processo legge un file, il contenuto dipende dalla temporizzazione delle scritture

Write on close

- Aggiornamento quando il file viene chiuso (semantica di sessione)
- Adatto per file aperti a lungo e modificati di frequente

Approccio centralizzato

- Client comunica apertura file f al server
- Server impedisce scritture se f aperto in lettura e qualunque operazione se f aperto in scrittura
- Semantica di tipo Unix
- Poca scalabilità

Caching vs. servizio remoto

- Caching
  - Riduce carico del server e traffico
  - Scalabile
  - Superiore nel caso di
    - Pattern d'accesso con scritture limitate
      - Inefficiente nel caso di molte scritture
    - Client con molta memoria e disco (fat client)
      - Inefficiente per diskless thin client

Replica di file

- DFS spesso fornisce repliche multiple di file
  - Affidabilità/Tolleranza ai guasti
    - "The show must go on"
  - Distribuzione del carico (prestazioni)
- Trasparenza
  - lo schema di naming deve rendere invisibile all'utente le repliche
  - Le repliche devono essere distinte tramite nomi "interni"

Replica di file

- Tipi di replica
  - Esplicita
    - A carico dell'utente
    - Server mantiene traccia degli indirizzi delle macchine dove esistono repliche
    - Quando si vuole aprire il file → Lookup del nome in sequenza sulle varie copie fino a trovarne una disponibile
  - Su domanda (lazy)
    - Il server copia asincronamente e in modo trasparente all'utente su altri server
    - E se il file viene modificato durante la copia?

Protocollo di modifica

- Aggiornamenti
  - Repliche = stessa entità logica ➡ aggiornamenti devono applicarsi a tutte le repliche
  - Aggiornamento di tutte le copie con messaggi sequenziali non fattibile → se il mittente cade a metà si ha inconsistenza
- Primary-copy replication
  - Un server primario, altri secondari
  - Aggiornamento solo rivolto al primario
    - Salva update su disco (per tolleranza a crash)
    - Aggiorna i secondari
  - Svantaggio: nel caso di crash di primario → no aggiornamenti
  - Soluzione: Voting

Protocollo di modifica

- Voting [Gifford 79]
  - Un client deve ottenere il benestare da un certo numero di server prima di accedere ad un file replicato
  - Ogni copia ha un numero di versione
  - Ogni scrittura aggiorna il numero di versione
  - Per capire quale replica è + aggiornata → Schema dei quorum
    - N repliche
    - Nr = read quorum
    - Nw = write quorum
    - Nr+Nw > N (almeno una delle repliche è aggiornata e non è possibile che due transazioni leggano e scrivano il file contemporaneamente)
    - Nr> N/2 (non è possibile che due transazioni scrivano il file contemporaneamente)

Protocollo di modifica

- Esempio
  - N=12
  - Nr=3 (es.: A,B,C)
  - Nw=10 (es.: C,…,L)
  - Almeno una (C) delle copie del quorum di lettura è aggiornata
  - In generale, il processo che apre il file può capire qual è la versione + aggiornata in base al numero di versione, almeno una delle copie del quorum è aggiornata

A	B	C	D
E	F	G	H
I	J	K	L

41

CASO DI STUDIO: NETWORK FILE SYSTEM (NFS)

42

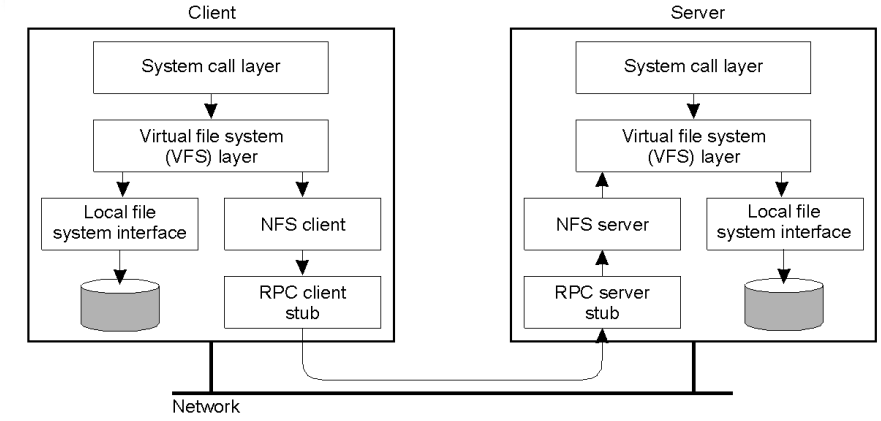
NFS

- Sviluppato da Sun
  - Inizialmente per piattaforme UNIX
  - Esteso ad altre piattaforme
- NFS è basato su un'architettura client/server, ma
  - la separazione tra client e server è concettuale
  - stesso SW in esecuzione
- Schema base
  - Server esporta directory
  - Client monta directory
  - Specifico protocollo che definisce le operazioni su file

Architettura di NFS

- Tre strati
  - System call layer
  - Virtual File System (VFS) layer
    - VFS mantiene una tabella di v-node (virtual i-node), uno per ogni file aperto
    - V-node puntano a
      - Un i-node nel S.O. locale
      - Un r-node nel client NFS
        - » Creato in seguito ad un'operazione di mount
  - NFS layer
    - Lato client e lato server
    - Traduce le richieste secondo il protocollo NFS

Architettura di NFS



45

Caratteristiche

- Stateful (a partire dalla versione 4)
  - Accesso tramite file handle autoconsistente
  - Server mantiene poche informazioni sulle connessioni aperte
- Pensato per semantica tipo UNIX
  - Supporta anche altri tipi
- Protocollo indipendente dal trasporto
  - Basato su RPC
- Locking
  - Non associabili a file aperti (server non sa quali file sono aperti!)
  - Necessari specifici meccanismi (per es. lock manager)
- Caching
  - Assume che la maggior parte dei file non sia condivisa
  - Basato su delayed write (write back)

46

Server NFS

- Server NFS permette la condivisione (export) di file system da parte dei client
- I FS esportati sono contenuti nel file /etc/exports
  - Esempio
    - /cdrom -ro host1 host2 host3
    - /home -alldirs 10.0.0.2 10.0.0.3 10.0.0.4
    - /a -maproot=root host.example.com box.example.org
  - Vedere man exports
- File di configurazione /etc/rc.conf
  - rpcbind_enable="YES"
    nfs_server_enable="YES"
    mountd_flags="-r"

47

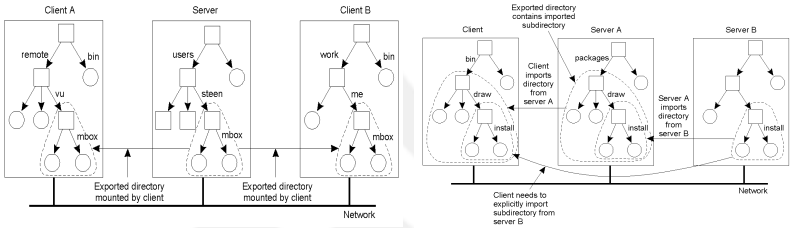
Server NFS

- Daemon attivi
  - nfsd (rpc.nfsd)
    - Accetta richieste dai client e le gira a mountd
    - Vedere man nfsd
  - mountd (rpc.mountd)
    - Gestisce le richieste che gli arrivano da nfsd
    - Vedere man mountd
  - rpcbind
    - Per consentire ai client di conoscere la porta su cui gira il server
    - Vedere man rpcbind

48

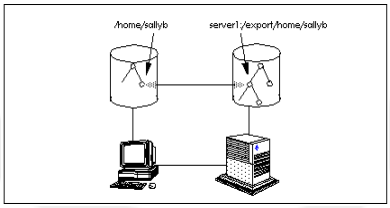
Client NFS

- Client NFS accedono ai file system del server facendo il mount dei FS esportati
  - mount server:path directory_di_mount
- Il risultato del mount è l'integrazione del FS remoto nell'albero locale
  - Possibilità di mount "annidati"
- Deamon: nfsiod (rpc.nfsiod)



Automounting

- Servizio addizionale del client
- Permette di montare/smontare FS in modo trasparente e su domanda
- Richiede mappe di automount
  - Associa nomi di client (mount point) con FS remoti
  - Mount points rappresentano lo spazio dei nomi del FS
  - Inserire il filesystem da montare nel file /etc/fstab
    - Vedere man fstab
- Esempio
  - server:/home /mnt nfs rw 0 0



Protocolli NFS

- Due protocolli basati su RPC
  - Per il mount
  - Per l'accesso a file

51

Protocollo NFS

- Protocollo di mount
  - Il client invia al server un messaggio (lookup) contenente un path
  - Messaggio non contiene il mount point
  - Se la directory corrispondente esiste ed è esportata (/etc/exports), il server ritorna un file handle
    - Tipo di FS/Disco/N° di inode/Info varie
    - Globalmente unico nel sistema
  - Accessi successivi a file nella directory montata usano questo handle

52

Protocollo NFS

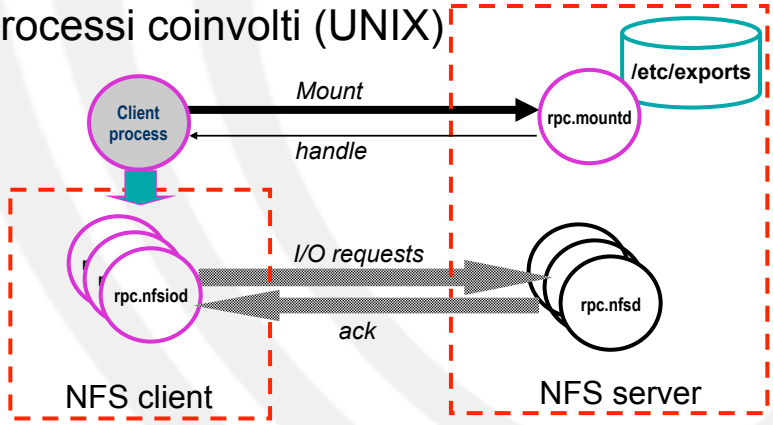
- Protocollo di accesso a file
  - Client invia richieste di accesso (tipo system call di UNIX)
    - Vedere slide successiva per esempio

Protocollo NFS – accesso a file

Operation	*v3*	*v4*	*Description*
Create	Yes	No	Create a regular file
Create	No	Yes	Create a nonregular file
Link	Yes	Yes	Create a hard link to a file
Symlink	Yes	No	Create a symbolic link to a file
Mkdir	Yes	No	Create a subdirectory in a given directory
Mknod	Yes	No	Create a special file
Rename	Yes	Yes	Change the name of a file
Rmdir	Yes	No	Remove an empty subdirectory from a directory
Open	No	Yes	Open a file
Close	No	Yes	Close a file
Lookup	Yes	Yes	Look up a file by means of a file name
Readdir	Yes	Yes	Read the entries in a directory
Readlink	Yes	Yes	Read the path name stored in a symbolic link
Getattr	Yes	Yes	Read the attribute values for a file
Setattr	Yes	Yes	Set one or more attribute values for a file
Read	Yes	Yes	Read the data contained in a file
Write	Yes	Yes	Write data to a file

Protocollo NFS
Schema Riassuntivo

- Processi coinvolti (UNIX)



Confronto riassuntivo

Issue	NFS	Coda	Plan 9	xFS	SFS
Design goals	Access transparency	High availability	Uniformity	Serverless system	Scalable security
Access model	Remote	Up/Download	Remote	Log-based	Remote
Communication	RPC	RPC	Special	Active msgs	RPC
Client process	Thin/Fat	Fat	Thin	Fat	Medium
Mount granularity	Directory	File system	File system	File system	Directory
Sharing sem.	UNIX/Session	Transactional	UNIX	UNIX	N/S
Cache consist.	write-back	write-back	write-through	write-back	write-back
Replication	Minimal	ROWA	None	Striping	None
Fault tolerance	Reliable comm.	Replication and caching	Reliable comm.	Striping	Reliable comm.
Recovery	Client-based	Reintegration	N/S	Checkpoint & write logs	N/S
Secure channels	Existing mechanisms	Needham-Schroeder	Needham-Schroeder	No pathnames	Self-cert.
Server groups	No	Yes	No	Yes	No
Name space	Per client	Global	Per process	Global	Global
File ID scope	File server	Global	Server	Global	File system
Access control	Many operations	Directory operations	UNIX based	UNIX based	NFS BASED