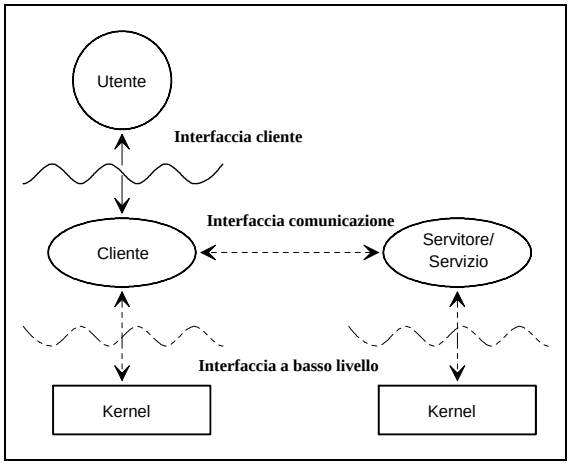
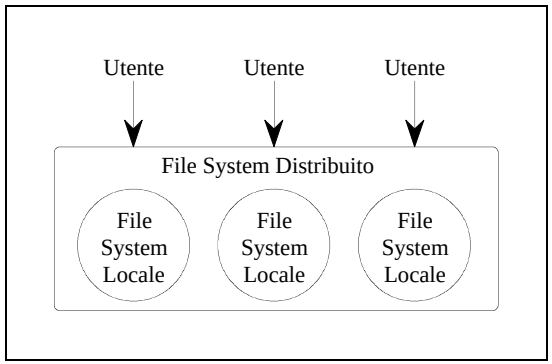
File System Distribuiti (F.S.D.)

Coordinamento delle operazioni sui file presenti su più nodi
File system in rete (**File system di Rete**)
File system Distribuiti (**FSD**)



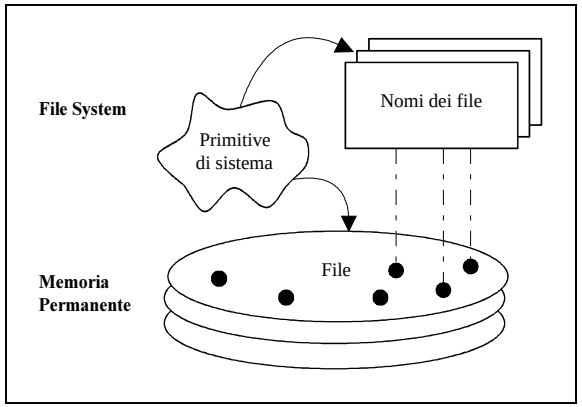
Modello Client-Server



Modello ad agenti coordinati

Ogni file system fa uso della ***memoria permanente***

- livello di nomi dei file
- livello fisico di accesso alle informazioni



livello monoutenza

Naming dei file ed interfaccia
Memorizzazione fisica dei file
Integrità dei file

livello concorrenza: supporto a più processi

Controllo della concorrenza
Serializzabilità
Deadlock

livello multiutenza

supporto al *time sharing*
problema della *sicurezza*

livello distribuito con proprietà

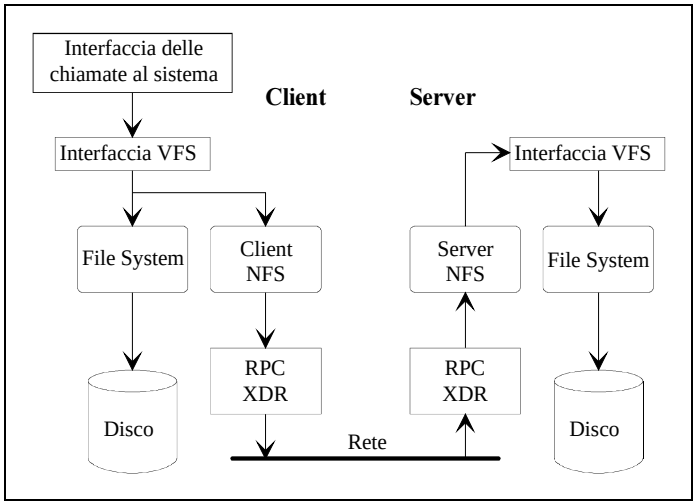
disponibilità
replicazione
implementazione efficiente e robusta

file system di rete *NFS*

Network File System

Obiettivi:

Eterogeneità
Trasparenza
Efficienza



Schema dell'architettura N.F.S.

Si realizza la *trasparenza* usando NFS tramite **Virtual File System** (VFS)

- separa le operazioni generiche sul file system dalla loro implementazione definendo un'interfaccia omogenea e trasparente, sia per i file locali che per quelli remoti
- utilizza la struttura *v-node*, con un identificatore numerico di file (simile all' *i-node* UNIX) unico in tutta la rete

NFS

Operazioni

- Visibilità iniziale ottenuta con ***mount***
- Accesso con operazioni (con **RPC**):
  - Ricerca di un file all'interno di una directory
  - Lettura di un set di nomi di directory
  - Manipolazione di link e directory
  - Accesso ad attributi di file
  - Lettura e scrittura di file

il server non ha stato (server *stateless*)

il server non ha informazioni sui file aperti dai clienti

Operazioni autocontenute di apertura, di lettura o scrittura del file all'interno di una singola chiamata RPC

Non ci sono operazioni di apertura e chiusura al server

- operazioni intrinsecamente **stateful**, e.g. lock di file, non possono essere implementate con tale protocollo
- operazioni come lettura e scrittura di direttorio possono entrare in conflitto con inserimenti e distruzioni

il client mantiene lo stato dell'interazione

il cliente ha una propria visione dei file globali che dipende solo dai mount fatti (e non è nota ad alcun altra entità)

Ogni operazioni di file (o direttorio) deve passare attraverso il cliente per la validazione: bisogna, nella ricerca di un nome assoluto tornare al client per ogni direttorio

La scelta stateless per ragioni di facile trattamento **guasti**
caso di **crash**

di un client nessuna ripercussione sul server
di un server nessun problema di recovery

Trasparenza per l'utente (?)

La semantica è la estensione della locale
ogni azione è propagata dal client al server per mantenere la massima consistenza
scelta molto costosa

NFS usa RPC (Remote Procedure Call) e (quindi) UDP
problemi nel caso di reti estese o attraverso router
mancanza controllo di congestione
Varianti NFS basate su TCP
Versioni NFS con lock

Comandi per il controllo del funzionamento di NFS:
nfsstat

Evoluzioni
automounting
montaggio by need al riferimento
replicazione (?) mancante

Implementativamente

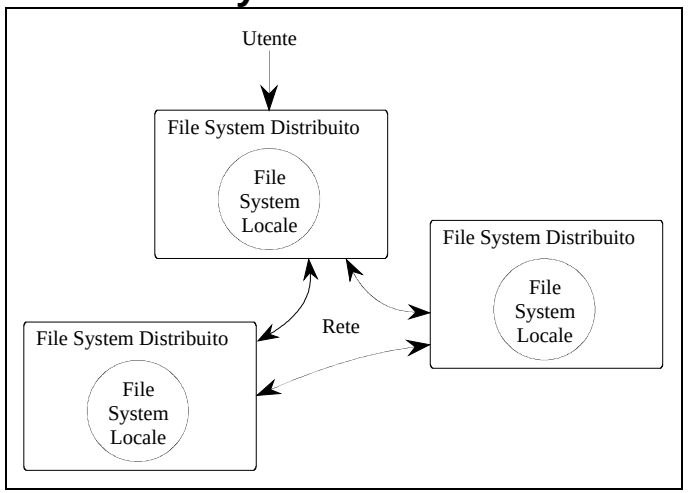
- Le versioni di NFS stabiliscono politiche per ottenere una migliore località
- Uso di demoni per attuare le richieste:

LATO **SERVER** composto di due demoni
- nfsd daemon
gestione delle richieste clienti: /etc/nfsd
si possono ottenere servizi paralleli attraverso una molteplicità di demoni attivi in attesa delle risposte
ogni server specifica anche il numero di processi server, secondo il servizio che si vuole fornire
- mountd attende le richieste di mount (controlla l'autorizzazione all'esportazione su /etc/exports) (possibili problemi di sicurezza)

LATO **CLIENT** composto da
- biod daemon
gestione della cache: /etc/biod gestisce la lettura a blocchi delle informazioni e il necessario trattamento di ottimizzazione
- automountd daemon
gestione dinamica del mounting: solo al riferimento di una parte di un direttorio che richiede un montaggio, il direttorio viene montato

Tipicamente, alcuni nfsd ed un automount sul server, un automount e alcuni biod sul server

File System Distribuito



maggiore coordinamento tra i diversi agenti

Proprietà di un File System Distribuito
Condivisione delle risorse e coordinamento

Ipotesi di località

Principio di località ==>

i riferimenti dei programmi tendono ad essere raggruppati entro zone limitate e locali, cioè un insieme limitato di pagine di memoria (*working set*)

Principio di località nel distribuito ==>

i riferimenti di applicazioni devono essere raggruppati entro domini limitati e locali, cioè un insieme limitato di nodi o di risorse logiche (*domain o scope*)

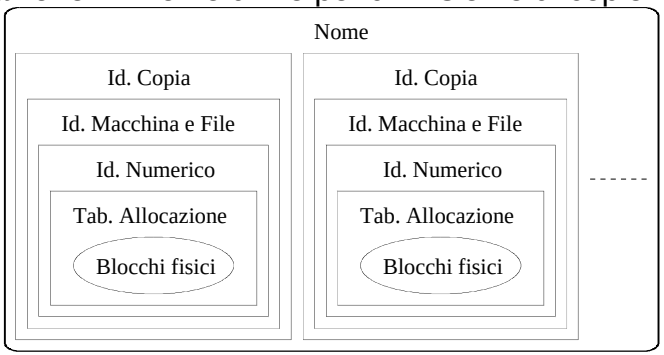
l'utente usa **più spesso risorse locali** (accesso veloce) e, **meno spesso**, risorse **non locali** (globali)

Naming e trasparenza

Gerarchia nel sistema dei nomi

uso di **nomi logici**, generalmente stringhe fino ai
uso di **blocchi fisici** nelle tabelle di allocazione su disco
F.S.D. aggiungono un ulteriore livello di astrazione
allocazione dei file nel sistema *trasparenza della rete*

Replicazione => nome di file per un insieme di copie



Nomi di gruppo sono collegati dal supporto alle diverse copie fisiche che sono presenti

Trasparenza allocazione

il nome del file non è correlato alla allocazione del file
==> *trasparenza dell'allocazione*

NFS nome **locale al cliente** (dopo il mount) **trasparente**
nome logico associato staticamente ad un nodo e corrispondente ad un diverso nome locale del servitore
==> ***condivisione dei dati*** tra utenti

Indipendenza dell'allocazione (caso riallocazione)

il nome del file non cambia neppure se varia la sua allocazione fisica
==> *indipendenza della allocazione*
mappa di allocazione dei file dinamica
dischi diversi in tempi diversi
(Locus e Sprite) nomi statici e indipendenti dalla allocazione della/delle copie fisiche

indipendenza allocazione ==> trasparenza allocazione

Indipendenza dalla allocazione

file spostati senza alterare il nome logico
mobilità o migrazione di file

gestione dei dispositivi
bilanciare occupazione dei dispositivi

Sistemi di naming

Strategie di *naming* dei file (sistemi di nomi)
COME È FATTO IL **SISTEMA DI NOMI**?

Naming gerarchico
Più semplice e meno trasparente
nome del file include un identificatore macchina di residenza **{*nomehost:nomefile*}**

non trasparenza né indipendenza dall'allocazione
(NFS fase di mount)

Naming a parziale condivisione
Aggiunta di una gerarchia remota allo spazio dei nomi logici locale (vedi UNIX *mount*)

trasparenza allocazione, non indipendenza allocazione

Naming a completa condivisone
totale integrazione di file system
tutti i client vedono la stessa struttura di nomi globale

indipendenza allocazione ==> trasparenza allocazione

Eccezione di alcuni file speciali (dispositivi ed eseguibili) che devono essere necessariamente locali

Semantica di operazione

La semantica specifica la proprietà delle modifiche apportate sui dati dai clienti e la visibilità ad altri clienti

accessi (in lettura e scrittura) tra *open* e *close ==> file session*

Semantica UNIX (compatibilità)

- ogni accesso in ***lettura*** al file vede gli effetti di ogni precedente scrittura
- ogni ***scrittura*** viene resa immediatamente visibile agli altri clienti che stanno accedendo allo stesso file

Possibile condivisione di I/O pointer per tutte le sessioni

Una sola immagine fisica: servizio secondo l'ordine di arrivo

Semantica a file condiviso immutabile

il file condivisibile non può essere modificato

- nome non riutilizzato
- contenuto non modificabile

Semantica di sessione (più immagini associate al file)

- ogni utente remoto accede ad una **copia** (lettura /scrittura)
- le modifiche di ogni utente remoto registrate alla **chiusura**
- i clienti locali eseguono modifiche immediate

Sessioni remote attive alla chiusura operano su copie obsolete

Semantica transazionale

transazione come sessione di accesso al file

- ogni utente remoto accede al file solo se non è in uso
- le modifiche vengono tutte eseguite, poi il file viene chiuso

Esecuzione serializzata delle transazioni

Uso di *lock* ==> limitazione del parallelismo

Dependability dei file

recupero (recovery) *robustezza* (robustness)
disponibilità (availability)

Robustezza reliability

Un file è *robusto,* o in *memoria stabile,* se è garantita la sopravvivenza delle informazioni anche in caso di guasto dei dischi

tecniche di *mirroring*, cioè replicazione su più dispositivi

Recupero recovery di situazioni di errore

Un file è *recoverable* se è possibile riportarlo, dopo un guasto, ad uno stato consistente precedente correlato ad altri file

Uso di protocolli di *commit*, e di *checkpoint*, ossia memorizzazione di stati precedenti ad istanti predefiniti

Disponibilità availability

un file è *available* se è sempre possibile l'accesso anche in caso di guasti

meccanismi di replica e di caching dei direttori

Un file robusto può non essere disponibile per un certo tempo fino al ripristino del relativo dispositivo

Un file robusto non è necessariamente recuperabile e viceversa

Replicazione dei file

meccanismi per replicazione tra nodi piuttosto che replicazione su dispositivi di macchina (*mirroring*)

requisito per replicazione

più copie di uno stesso file su macchine *failure-independent*

la disponibilità di una replica non deve influire sulle altre

Gestione delle modifiche

consistenza vs. disponibilità

Tolleranza ai guasti

USO dello STATO

Servizi stateful e stateless

Un server può mantenere informazioni sulla connessione ed i servizi che sta offrendo ai cliente ==> *stateful*
Se il server non memorizza alcuno stato ==> *stateless*

Server stateful

il server mantiene un identificatore univoco per ogni cliente
- Accesso veloce a *circuito virtuale* tra cliente e server
- Migliore performance

In caso di **guasti**
server le informazioni del server sono perse
recupero dello stato con dialogo con i clienti o
abort di tutte le operazioni in corso
client il server deve liberare la memoria occupata da informazioni di un client perduto

Server stateless

il server non mantiene informazioni dei cliente
- ogni servizio è *indipendente* dal precedente e *idempotente*
- operazioni di collegamento sono ridondanti

robustezza ai guasti
alta ridondanza operazioni e informazioni sulla rete

Scalabilità

Definizione
- *Scalabilità* come la possibilità di espandere un sistema senza causare un degrado di prestazioni

Un **sistema scalabile** raggiunge la **saturazione** più tardi di un equivalente **non scalabile**

Ipotesi di località
Ipotesi di *bounded resource* (uso di risorse limitate)
- richiesta di servizi di ogni componente del sistema limitata da una costante indipendente dal numero di nodi del sistema

Il principio delle **risorse limitate** per canali e traffico di rete ==> sconsiglia messaggi broadcast

meccanismi di caching, hint, migliorano **scalabilità**

semantiche di condivisione riducono **scalabilità**
controllori o risorse centralizzate riducono **scalabilità**

configurazioni funzionalmente **simmetriche**
ogni macchina ha un proprio grado di autonomia ed un uguale ruolo all'interno del sistema

Cluster

Configurazione simmetrica ed autonoma
- metodo del *clustering* (ossia di **località**)

cluster come set di macchine interconnesse ad un server
- minimizzazione accessi intercluster
- massimizzazione operazioni intra-cluster

Problemi di implementazione

Tecniche di ricerca dei file

*trasparenza dell'**allocazione*** dei file ==>

conversione del nome logico in nome di basso livello con COSTO, EFFICIENZA

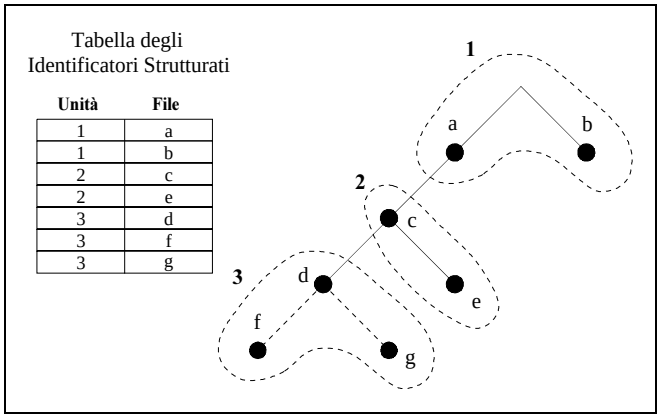
Tecnica di ricerca di *lookup ricorsivo*

il percorso specificato dal nome del file viene passato ad un server che lo risolve passandolo a sua volta eventualmente ad altri server

Identificatori strutturati

tabelle mantenute in memoria dal cliente per associare ad un passo solo i nomi logici dei file alla allocazione fisica ==> *identificatori strutturati*

- la prima parte identifica la *unità componente*, cioè il server di allocazione del file,
- la seconda identifica il file all'interno del server



Hint (Consigli nell'Accesso)

la ricerca di un file tramite *hint* cioé attraverso un *semplice consiglio di ricerca* ==>

le informazioni di allocazione ottenute con un precedente accesso al file possono non essere più valide

Spesso le indicazioni sono puramente orientative, anche se è bene che valgano per intervalli lunghi (pochi errori)

caso di **NFS**

un cammino può avere portato ad un certo file in un certo file system, ma il file stesso potrebbe non essere più diretto verso il corrispondente fisico trovato

- se il cambiamento è causato dal cliente stesso, il livello cliente può averne traccia
- se il cambiamento è causato dal file system del server, si deve ottenere la nuova informazione dal server stesso

Ogni direttorio remoto deve (idealmente) essere continuamente validato

Accesso a file remoti

Servizio remoto vs. caching

Per il trasferimento dati due metodi complementari

- *servizio remoto (richiesta a nodo remoto)*
- *caching (mobilità e copia)*

Servizio remoto

Ogni richiesta di accesso a file viene inviata al server, questo esegue l'operazione richiesta e invia il risultato al client

corrispondenza tra accessi e traffico dal/al server

Problemi con alto traffico

Caching

tecnica di caching (o buffering), memorizzazione temporanea in memoria di blocchi di dati da disco

cache del server (vedi caso locale)

cache del cliente ==>

riduzione del carico del server e del traffico generato

usato anche in combinazione con remote service

località dei riferimenti

Il file è la **copia principale** del server

se esistono più copie del file nelle cache clienti, le modifiche di una cache deve essere propagata al file server originario e di qui alle altre copie

problemi di *consistenza cache*

Convalida della cache

decisione se la propria copia è consistente con la principale
In caso di inconsistenza ==>

protocollo di aggiornamento

client-initiated approach
server-initiated approach

Client-initiated approach

Il cliente controlla la consistenza delle informazioni con richiesta al server

frequenza di controllo

da un controllo per ogni accesso alla cache fino ad un solo controllo all'apertura del file

Alternativamente il controllo richiesto periodicamente

Controllo introduce ritardo nelle richiesta di accesso
Evitare un eccessivo traffico in rete

Server-initiated approach (server con stato)

Il server controlla le copie di ciascun cliente

Se situazioni di potenziale inconsistenza dei dati, richiede l'aggiornamento

necessità di mantenere nozione dei clienti correnti

con ***semantica di sessione***, ad una richiesta di chiusura di un file modificato, il server avvisa i clienti dell'invalidità delle copie

Tecniche di modifica della cache

rispetto alla copia principale

write through*, *delayed write*, *write on close* e *periodic write

Write through

invio immediato dei dati modificati al server ed a tutti i client per aggiornare le cache

Affidabilità ma scarsa efficienza

Delayed write (scrittura ritardata)

modifiche scritte nella cache locale e solo più tardi propagate al server e alle altre copie

Migliore performance, problemi di affidabilità e consistenza

Write on close

scrittura alla chiusura, cioè aggiornamento solo alla chiusura del file

ottime performance in accessi prolungati con frequenti modifiche ad un file, problemi di consistenza ed affidabilità

Periodic write

esame periodico della cache e aggiornamento solo dei blocchi modificati

problemi di consistenza ed affidabilità dipendenti dal periodo

Realizzazione della cache

Dimensioni della cache

Granularità del dati contenuti nella cache variabile

Tecnica di *read-ahead*

client e server bufferizzano i dati ancora prima che questi diventino necessari

aumento della **dimensione** di unità di cache ==>
aumenta l'**efficacia** read-ahead
tempo richiesto per il trasferimento dei dati
probabilità di problemi di **inconsistenza**

Allocazione della cache

cache in **memoria centrale** o su **disco**

primo caso

veloce accesso ai dati
anche per macchine diskless

secondo caso

maggiore tempo di accesso ai dati
maggiori dimensioni ed affidabilità della cache

Politiche di flushing

blocco sporco (*dirty block*) dati, in cache, modificati e non più consistente con la copia primaria
flushing azione di scrittura di dirty block sulla copia principale

Problemi di aggiornamento e di consistenza

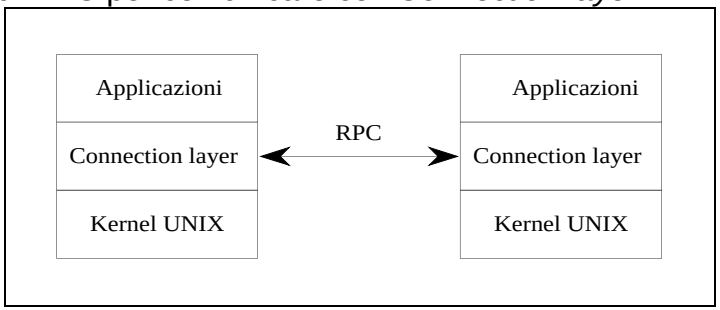
UNIX United

UNIX United una delle prime estensioni a UNIX
strato software, detto *Newcastle Connection layer*, posto tra le applicazioni ed il kernel UNIX

Unica struttura
ogni sistema componente è un directory
== > *trasparenza della rete*
non trasparenza alla allocazione
buon livello di autonomia ed espandibile
nessuna replicazione delle copie ed affidabilità

Supporto non molto efficiente

Uso di RPC per comunicare con *Connection layer*



Ricerca di un file ==> su tutte le macchine del percorso
il sistema mantiene informazioni sull'indirizzo e il routing per accedere ad ogni *file descriptor*
connessioni di tipo ***stateful***

Network File System (Sun)

Network File System
per un numero limitato di workstation in rete
integrato nel kernel SUN OS per efficienza
ma anche realizzazioni al di fuori

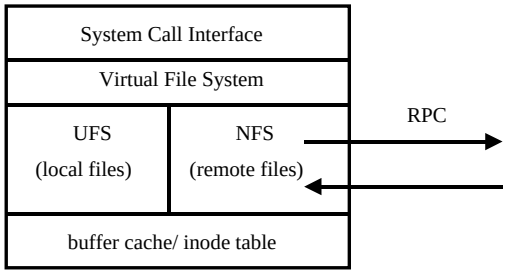
L'obiettivo
condivisione di file in maniera trasparente
ma efficiente

Meccanismo del *mount* =>
trasparenza alla allocazione del file
non indipendente dall'allocazione
Naming a parziale condivisione
cliente responsabile dei nomi:
il cliente non vede ciò che il server ha montato ma solo il file system locale del server
il cliente può montare su parti di file system che sono già derivati da server (cascade mount)

Virtual F.S. usa struttura chiamata *v-node*
mount table identifica *macchina* ed *i-node*
uso di RPC
accessi al server di **tipo stateless** (senza *locking*)

Implementazione di NFS

Il **v-node** specifica la necessità di passare ad un altro sistema locale



Ogni **singola operazione** viene richiesta attraverso **RPC**
(uso di UDP e opzionalmente TCP, 2049)

RPC (versione 3) consente di:

- **getattr, setattr, statfs**
- **lookup**
- **read/write**
- **creat, remove, rename**
- **link**
- **symlink, readlink**
- **mkdir, rmdir**
- **readdir**

mancano operazioni di stato (tipo lock)
cosa succede con operazioni concorrenti

NFS usa due meccanismi di cache del cliente a tempo
file, a memorizzazione locale di file (3 sec.)
file attribute, (direttori 30 sec.)
più informazioni sui server e sugli i-node virtuali utilizzati recentemente

Locus File System

LOCUS estende il kernel di UNIX per realizzare un
file system globale
con proprietà di **tolleranza ai guasti**
ogni macchina ha una vista completa e consistente del F.S. virtuale (unico sisitema di nomi)

La struttura dei file logici
===> ***replicazione***
trasparenza **e** ***indipendenza*** **dall'allocazione**

Unità componente ***filegroup*** **logico**
formato da un insieme di copie di file fisici

in astratto, una *struttura unica*
in concreto, fisicamente mappato in *multiple file container* memorizzati e replicati in maniera trasparente

file designator indicatore a basso livello del file
{numero di filegroup, numero di i-node}
indipendente dalla allocazione

File system a completa condivisione

Il sistema mantiene una **tabella** in cui per ogni filegroup
file designator della directory logica corrente e
nodo per l'accesso sincronizzato nel filegroup

ACCESSO

Accesso ai file in LOCUS coinvolge tre entità
su macchine diverse
using site o **us**, richiede l'accesso al file (client)
storage site o **ss**, fornisce i dati (server)
current synchronization site *o* **css**, per gestire la sincronizzazione nell'accesso al file
sceglie **ss** per le operazioni di apertura del file e mantiene informazioni di consistenza delle repliche

Al primo accesso, **css** cerca negli **ss** la versione più aggiornata e la passa allo **us**

Operazioni di lettura accedono da **us** ad **ss**

Totale trasparenza dell'allocazione
Tecniche di *read-ahead* e *caching*
sia di pagine dati sia di percorso
Connessione tipo stateful
sia **ss** e **css** mantengono informazioni sui file aperti

Operazioni in scrittura
css seleziona una copia primaria, *primary copy*, in cui registra le modifiche con *delayed write*

LOCUS implementazione

LOCUS utilizza il meccanismo di *shadow page ==>*
atomicità operazioni di scrittura
consistenza dei dati
maggiore tolleranza ai guasti
Modifiche non apportate al file, ma registrate
In caso di aborto vengono scartate,
altrimenti vanno a sostituire quelle obsolete nel file

In caso di modifica, **css** avverte gli altri **css** ed **ss**

LOCUS
estende sincronizzazione UNIX in distribuito
politica tipo *exclusive-writer-multiple-readers*
un solo processo scrive mentre altri lo possono leggere

Anche strumenti per l'accesso esclusivo a file
semantica di tipo transazionale

Possibilità di lavorare anche in caso di
partizione del sistema *(network partition)*

SUPPORTO ad-hoc

Uso di funzioni di comunicazione integrate nel kernel
Il supporto avviene con processi leggeri interni al kernel attivati per ogni richiesta di file
meccanismi ad-hoc efficiente

Sprite

SPRITE è un sistema operativo distribuito sperimentale basato su workstation con alta capacità di memoria e cache (ma spesso senza disco locale)
Necessità di caricare e mantenere in memoria interi file system (ipotesi 500Mbyte di memoria)

Unica struttura UNIX
==> **trasparenza dell'allocazione**
mantenendo **efficienza** della **implementazione**

SPRITE kernel
Integrazione della memoria virtuale con il FS
memoria virtuale realizzata con file, *backing file*
uso di thread come processi leggeri
per ottenere il bilanciamento uso di
migrazione dei processi e dei file

Naming a parziale condivisione
Unico file system logico suddiviso su più *domini*, ossia sottoalberi memorizzati interamente su più server
Ogni macchina mantiene mappa dei server, *prefix table*, per raggiungere la directory di più alto livello di ogni domain
Il client seleziona il server dalla propria tabella
se la richiesta fallisce, il client manda un messaggio *broadcast* con il nome del file
Il server con il file risponde con il proprio prefix, ed il client aggiorna la *table*

SPRITE
uso di hint rappresentato dalla tabella
uso di messaggi broadcast
limita di scalabilità

SPRITE

semantiche di condivisione UNIX
metodo ibrido per convalida cache
Apertura in scrittura incrementa la versione del file
Il client confronta la propria versione con il server
se differenti, il client aggiorna il file e lo ricarica dal server

in caso di operazioni di sola lettura, si possono avere molte copie locali mantenute vicine ai clienti

server di tipo stateful
se il server ha memorizzato un file obsoleto, aperto in scrittura da un utente che non ha ancora aggiornato la copia, forza il *flushing*
Se il file è aperto in scrittura da più utenti
viene disabilitato il caching ed è necessario mantenere un migliore aggiornamento delle copie

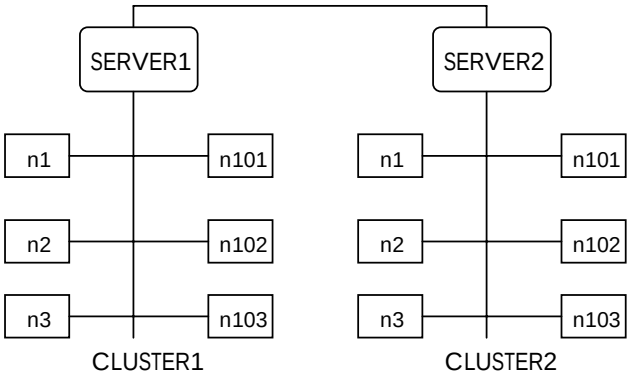
Andrew File System (AFS)

ANDREW è un ambiente di calcolo distribuito
per realizzazione di grandi sistemi fortemente scalabili

Ipotesi di architettura

Macchine client e server

Client workstation con spazi file ***locale*** e ***condiviso***
Server dedicati alla memorizzazione trasparente dello ***spazio condiviso***



Per scalabilità ==> cluster di workstation e server collegati tramite un server detto *router* al *backbone*
rete di maggiori dimensioni

In genere, si privilegia l'accesso a risorse locali: ogni cliente vede il proprio spazio locale, o, in alternativa lo spazio del server proprio come condiviso
la **località** viene attuata con il movimento dei file
(o ***blocchi di file***)

Indipendenza dell'allocazione dei file

naming a completa condivisione

Spazio condiviso costituito da unità, o *volumi*,
di dimensioni limitate, unite tra loro
tramite un meccanismo simile al mount
Informazioni di allocazione replicate su ogni server in un *volume location database* cui accedono i clienti

I volumi possono migrare, in modo atomico
il server originario continua a gestire gli accessi al volume spostato ancora per un certo tempo

Andrew
semantica di sessione condivisa

Duplicazione locale di ogni file aperto
si interagisce con il server solo alla apertura/chiusura
Ogni modifica di un file non visibile fino alla chiusura
Il cliente lavora in locale a meno di call back del server

Cache velocizza **operazioni di apertura**
file in cache valido fino alla invalidazione da parte del server (stateful)

Problemi **in ripristino** in caso di guasti

ANDREW definito intermini di processi leggeri
server prevedono processi leggeri
con *nonpreemptive scheduling*
per soddisfare le richieste dei client

Estensioni possibili

Nuovi sistemi basati su **architetture globali** e su **condivisione globale** delle risorse di memoria

OceanStore

sistema cooperativo di gestione di informazioni

alta availability

da ovunque **accesso ubiquo**

Implementazione

confederazione di sistemi di supporto che si organizzano attraverso scambi di servizi e di uso risorse

- **Ogni oggetto ha un ID unico**

GUID globally unique Identifier

- **Replicazione su molti server diversi in zone diverse**

le copie non sono ad allocazione fissa ma variaible e si possono muovere **(repliche floating)**

- **Ogni variazione crea una nuova versione**

Scritture delle informazioni (operazioni di cambiamento di parte) producono versioni che tendono a coesistere

- **Oggetti sono presenti in forma attiva o di archivio**

Le forme archiviate sono diffuse con maggiore grado di replicazione

- **Controllo degli Accessi sugli oggetti**

si possono restringere gli accessi sia in lettura, sia in scrittura

Sistemi P2P e non Cliente/Servitore

I nuovi sistemi si basano su un modello qualificato come **Peer-to-Peer (P2P)**

Non esiste un **server unico**, ma una serie di località diverse che possono contenere copie delle informazioni

Il lavoro quindi avviene dalla cooperazione dei diversi pari che tendono a produrre una **visibilità globale** attraverso una **visione** e **operazioni locali**

Molto lavoro recente:

Freenet

file system (come OceanStore) con elevato grado di replicazione ed a diffusione derivante dalle richieste

strategie di propagazione ispirate al routing globale

sistema di nomi basato su indici (integrità via hash)

Publius

condivisione di informazioni con diversi gradi di sicurezza e privatezza

SETI@home

applicazione di ricerca ed analisi di segnali estraterrestri eseguita su macchine clienti

gnutella

condivisione di informazioni musicali

Jabber

supporto alla conversazione e cooperazione tra utenti

ecc. ecc.