



**Università degli Studi di Bologna
Facoltà di Ingegneria**

**Corso di
Reti di Calcolatori L-A**

File System Distribuiti

Antonio Corradi

Anno accademico 2009/2010

SISTEMI OPERATIVI di RETE e DISTRIBUITI

**Network Operating Systems (NOS) vs.
Distributed Operating Systems (DOS)**

I NOS indipendenti e non trasparenti

I DOS coordinati e trasparenti e aperti

paralleli al loro interno - uso di risorse parallele interne

paralleli a livello di utente - con maggior omogeneità

capaci di gestire nuove risorse non note da aggiungere alle
statiche

? La trasparenza consente di modificare il sistema ?

**Si deve avere sia trasparenza a livello utente sia visibilità a livello di
sistema - PUNTI di VISTA DIVERSI e multipli**

**allo stato dell'arte i sistemi devono fornire anche visibilità e livelli
diversi di trasparenza per diversi livelli di uso**

SISTEMI OPERATIVI DISTRIBUITI

I DOS e NOS ottimizzano l'uso delle risorse distribuite sulla base della CONDIVISIONE per

- **scambio e replicazione delle informazioni**
- **parallelismo nella computazione**
- **ridistribuzione del carico**

Unica macchina virtuale con proprietà che mirano al:

Controllo allocazione delle risorse, gestione della Comunicazione, Autorizzazione, Trasparenza (qualche livello), Controllo dei servizi del sistema (QoS), Capacità evolutiva (dinamicità)

Apertura del sistema (OPEN system)

le risorse possono essere inserite durante la esecuzione del sistema e non sono note staticamente prima della esecuzione

Naturalmente, non esistono sistemi aperti in assoluto, ma aperti a fronte di alcune variazioni e di alcuni cambiamenti (previsti)

SISTEMI di RISORSE nel DISTRIBUITO

SISTEMI OPERATIVI DISTRIBUITI come insieme di gestori di risorse - **obiettivo Resource management**

Processor management, Process management

Memory management, **File management**

Molti i filoni coinvolti come la gestione di servizi diversi:

- Replicazione risorse, Gestione dei Nomi, Processi

- Comunicazione e Sincronizzazione, Sicurezza

- Allocazione e riallocazione delle risorse

- Gestione dei Servizi applicativi e della QoS

Importante la standardizzazione delle soluzioni

INDIPENDENZA dalla ARCHITETTURA

APERTURA (OPEN SYSTEM)

INFORMAZIONI SUL SISTEMA (MONITORAGGIO)

OVERHEAD di SISTEMA

TREND overhead

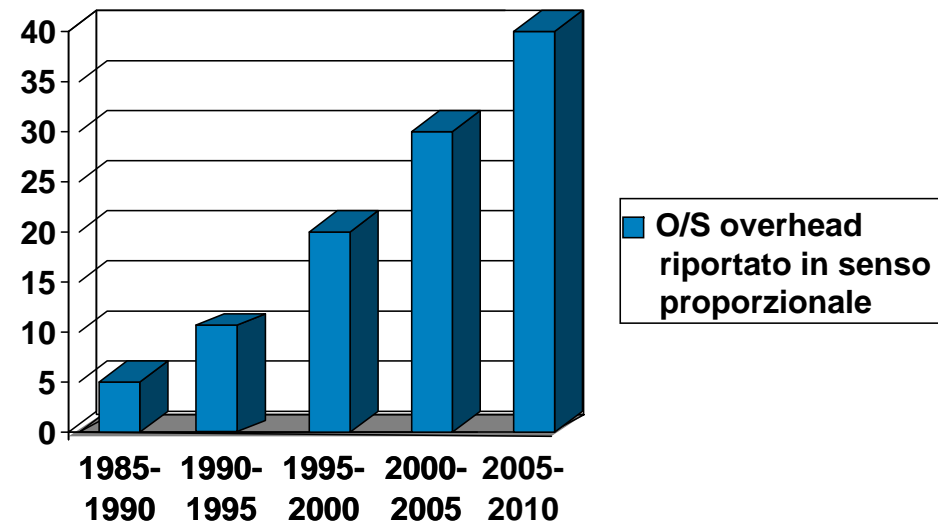
prestazioni in aumento con nuove tecnologie o miglioramenti dell'esistente: sia banda sia latenza

Anche l'Overhead della parte di sistema operativo, cioè della gestione locale, aumenta in proporzione

Anche se ci possiamo aspettare miglioramenti in prestazione delle diverse parti di un sistema

(comunicazione, dischi remoti, accessi veloci, etc.)

Il problema da risolvere è la corretta gestione

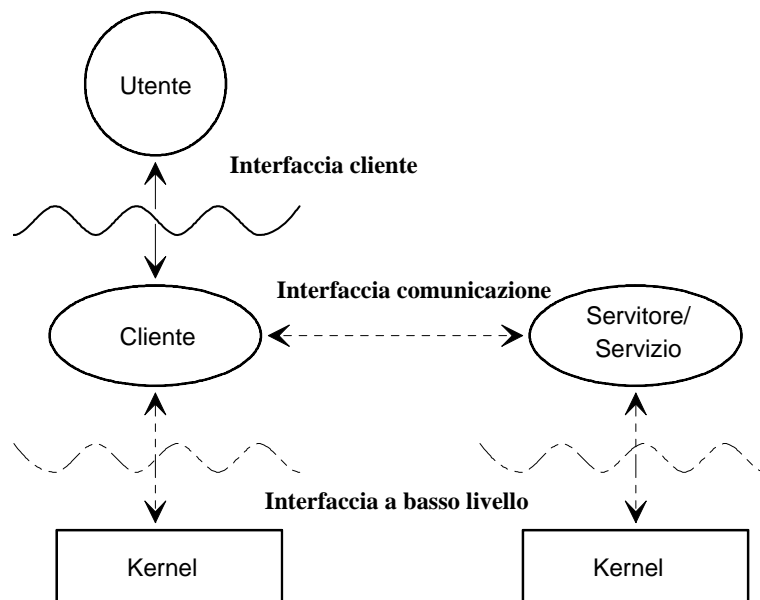


File System Distribuiti (F.S.D.)

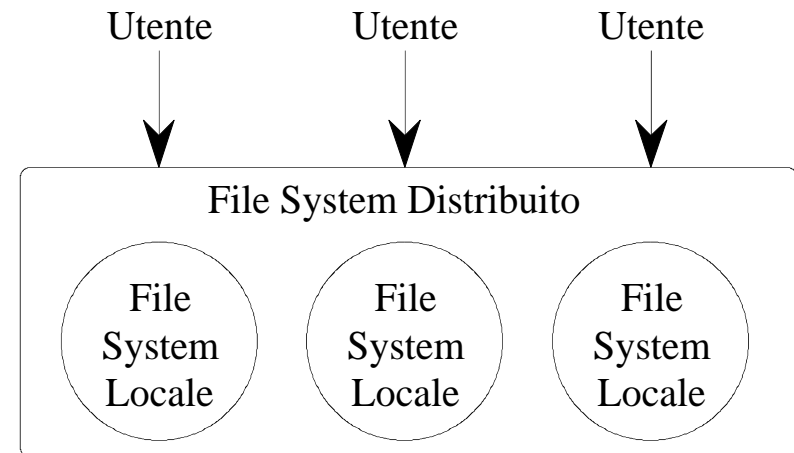
Coordinamento delle operazioni sui file presenti su più nodi

File system in rete (File system di Rete)

File system Distribuiti (FSD)



file system di rete



file system distribuiti

FILE SYSTEM a LIVELLI

Ogni file system fa uso della **memoria permanente** per il livello di nomi dei file e il livello fisico di accesso alle informazioni

livello monoutenza: file fisici e logici

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

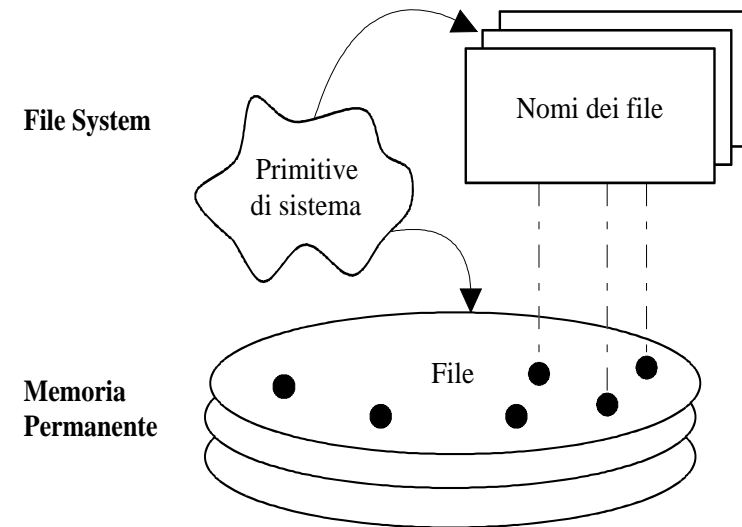
livello concorrenza: supporto a processi

- Controllo della concorrenza
- Serializzabilità e Deadlock

livello multiutenza: più utenti

- supporto al time sharing
- problema della sicurezza

livello distribuito con proprietà come disponibilità (QoS) via replicazione ed implementazione efficiente e robusta



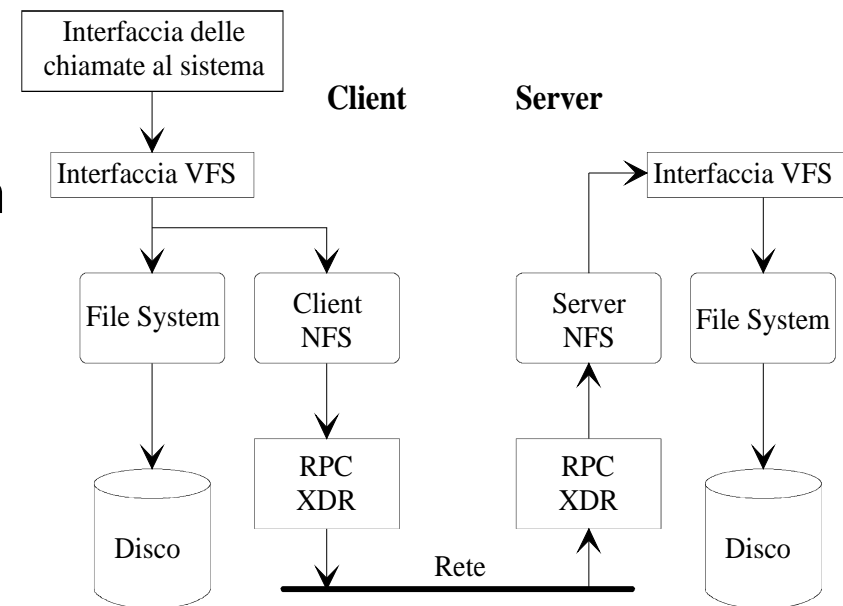
NETWORK FILE SYSTEM (NFS)

Il Network File System o NFS di SUN, come file system di rete

Obiettivi del progetto: **Eterogeneità, Trasparenza, Efficienza**

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

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



Schema dell'architettura NFS

NFS: PRINCIPI

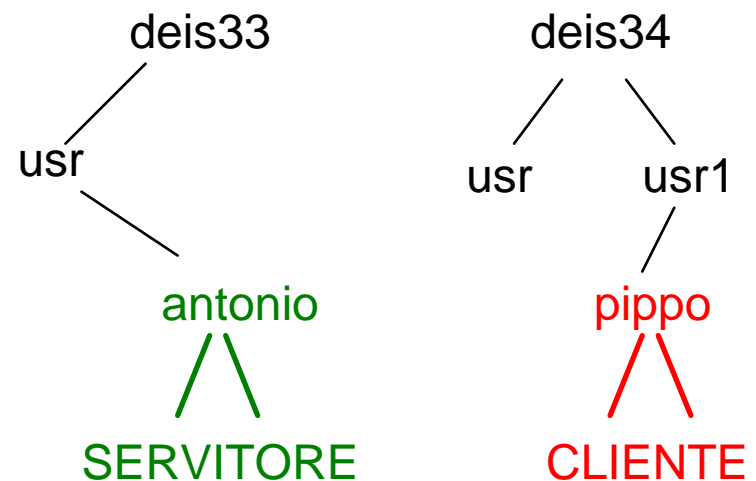
Network File System come **Sistema Distribuito per UNIX SUNOS**
senza TRASPARENZA alla ALLOCAZIONE

- * il superutente del server (deis33) deve autorizzare l'esportazione (/etc/exports) del file system richiesto
- * il superutente del client (deis34) deve montare (mount) il file system di deis33 **mount deis33:/usr/antonio /usr1/pippo**

Dopo il mount, tutta la gerarchia /usr/antonio è visibile agli utenti del client come /usr1/pippo

da deis34 si vedono i file di deis33
e scompare la visibilità del direttorio locale

Non sono ammessi **montaggi innestati**
ma un cliente **può montare nuovi direttori**
su direttori già montati
(conosce sempre localmente il server)



NFS: TRASPARENZA e COSTO

TRASPARENZA nella COMUNICAZIONE

a livello utente si vedono i file come se fossero LOCALI

Ogni file system può contenere parti degli altri

I file relativi sono condivisi tra diverse macchine

(con visibilità sempre locali e senza un sistema di nomi consistente)

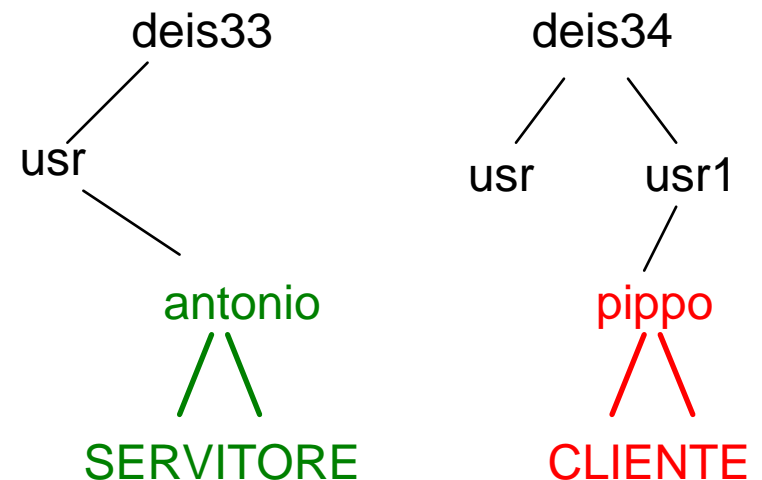
COSTO ed EFFICIENZA

In genere, si tende a fornire **semantica UNIX**,

cioè propagazione di ogni cambiamento agli altri utenti del file

Ogni cambiamento fatto su un file dovrebbe essere propagato dal servitore al cliente (se si conoscono?)

Miglioramento attraverso replicazione via cache (che snaturano la semantica)



NFS: OPERAZIONI POSSIBILI

Le **operazioni possibili di NFS** sono:

- * Accresciuta Visibilità iniziale, ottenuta con mount
- * Accesso via operazioni remote (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 NFS non ha stato (server stateless)

per questione di costi e di efficienza

il server non ha informazioni sui file aperti dai clienti

Operazioni sui file remoti, di lettura o scrittura del file, espresse come una chiamata RPC

NFS: SERVER STATELESS

Il server NFS è senza stato:

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

il client deve mantenere lo stato dell'interazione (cioè le sessioni aperte sui file che conosce solo lui via v-node)

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 server per la validazione: bisogna tornare al server per ogni direttorio, nella ricerca di un nome assoluto

La scelta stateless permette un più facile recovery in caso di crash

di un client

nessuna ripercussione sul server

di un server

nessun problema di recovery

PROPRIETÀ NFS

Trasparenza per l'utente (?)

La **semantica UNIX** è (dovrebbe essere?) la estensione della locale ed è una scelta molto costosa

ogni azione è propagata dal client al server per mantenere la massima consistenza

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, ossia montaggio by need al primo riferimento di un file del server

La replicazione è sempre mancante e non trattata

DEPLOYMENT NFS: i DEMONI

Implementativamente, le versioni di NFS stabiliscono **politiche per favorire una migliore località**, in particolare attraverso l'uso di demoni ottimizzati per servire le richieste

- **nfsd daemon - LATO SERVER**

gestione delle richieste clienti con servizi paralleli attraverso una molteplicità di demoni attivi in attesa delle risposte (# servitori)

- **mountd daemon - LATO SERVER**

attende le richieste di mount (controlla l'autorizzazione all'esportazione su /etc/exports) (soggetto a possibili problemi di sicurezza)

- **biod daemon - LATO CLIENT**

gestione della cache con la lettura a blocchi delle informazioni e il necessario trattamento di ottimizzazione (# clienti)

- **automountd daemon - LATO CLIENT**

gestione dinamica del mounting: solo al riferimento di una parte di un direttorio che richiede un montaggio, il direttorio viene montato

FILE SYSTEM DISTRIBUITI

Una proprietà per la implementazione dei File System Distribuiti per la condivisione delle risorse e coordinamento

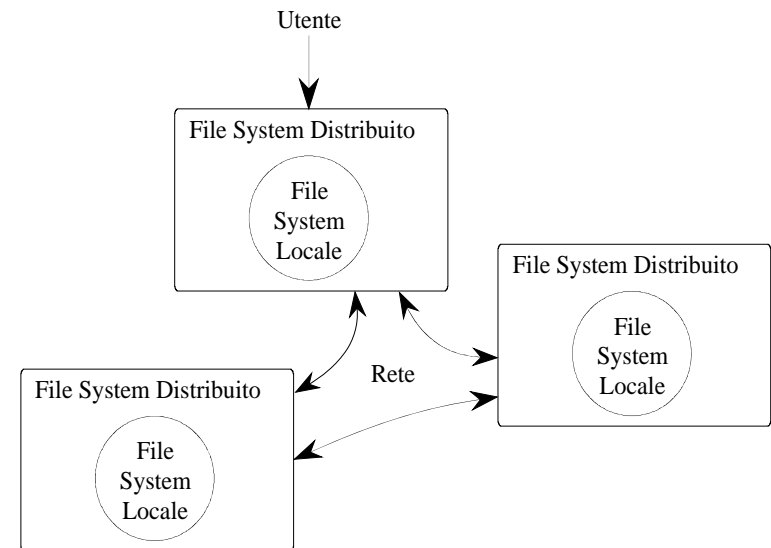
Principio di località

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 locali, cioè su un insieme limitato di risorse logiche

Un utente accorto usa più spesso risorse locali (accesso veloce) e, meno spesso, risorse non locali (globali) se vuole ottenere un supporto efficiente



SCALABILITÀ

Scalabilità come possibilità di espandere un sistema a piacere **senza causare degrado in prestazioni**

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

In genere, non esistono sistemi scalabili in generale

Tutti i sistemi ottengono prestazioni che dipendono dal numero delle entità partecipanti

la crescita del numero comporta delle variazioni avvertibili in prestazioni

Ipotesi di **località o 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

Se rispettata la località, \Rightarrow allora si ottiene scalabilità

Imponiamo dei vincoli sulla interazione, e non sui partecipanti ...

SCALABILITÀ

Vincoli di località ⇒ per ottenere un range di scalabilità

Il principio delle risorse limitate per canali e traffico di rete

- suggerisce meccanismi di caching per scalabilità
- configurazioni funzionalmente simmetriche con ogni macchina con un proprio grado di autonomia ed un ruolo pari all'interno del sistema
- sconsiglia messaggi broadcast
- sconsiglia semantiche di condivisione
- sconsiglia controllori o risorse centralizzate

In caso architetturale di processori si introducono **Cluster** come configurazione simmetrica ed autonoma di processori organizzati in una limitata località

cluster come set di macchine interconnesse, cercando di minimizzare gli accessi intercluster e di massimizzare tutte le operazioni intra-cluster, favorendo la località e limitando le operazioni costose

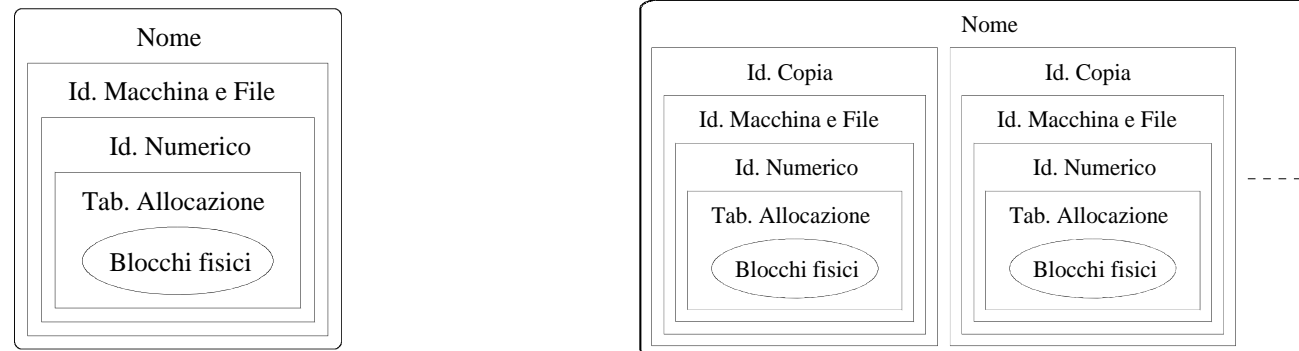
SISTEMI di NOMI

Sistemi di Nomi, replicazione, e trasparenza

Gerarchia nel sistema dei nomi

uso di nomi logici, generalmente stringhe fino all'uso di blocchi fisici
nelle tabelle di allocazione su disco

FSD aggiungono un ulteriore livello di astrazione per tenere traccia della allocazione dei file nel sistema distribuito e con trasparenza della rete



Nomi di gruppo collegati dal supporto alle diverse copie fisiche che sono presenti per supportare la replicazione: nome di file per la replicazione sull'insieme di copie

IN / VISIBILITÀ della ALLOCAZIONE

La **trasparenza all'allocazione** permette che il nome del file non sia correlato alla allocazione del file e non ne dia visibilità

In NFS, il nome locale del cliente è trasparente (dopo il mount)

anche se il nome logico associato ad un nodo è corrispondente ad un diverso nome locale del servitore condivisione dei dati tra utenti

Indipendenza dell'allocazione (in riallocazione)

il nome del file non cambia neppure se varia la sua allocazione fisica
mappa dinamica di allocazione dei file

(Locus e Sprite) nomi statici e indipendenti dalla allocazione della/delle copie fisiche

indipendenza allocazione vs. trasparenza allocazione

Indipendenza permette che i file siano mossi di allocazione senza alterare il nome logico, ad esempio in caso di mobilità di file

gestione dei dispositivi bilanciare occupazione dei dispositivi

SISTEMI di NOMI dei FILE

Strategie di naming dei file (e sistemi di nomi per i file system)

Naming gerarchico

Politica più semplice e non trasparente: il **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 **gerarchie remote allo spazio dei nomi logici locale**
(vedi UNIX dopo il mount)

trasparenza allocazione, non indipendenza allocazione

Naming a completa condivisione

totale integrazione del file system **per tutti i client che vedono la stessa struttura di nomi globale in modo uniforme**

indipendenza allocazione trasparenza allocazione

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

SEMANTICA delle OPERAZIONI

La semantica specifica la **proprietà delle modifiche** apportate sui dati dai clienti in relazione alla loro visibilità ad altri clienti

Gli accessi considerati sono letture e scritture nella sessione sul file, delimitata ed identificata da open/close

Semantica UNIX (compatibilità)

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

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

Possibile condivisione di I/O pointer per tutte le sessioni

Semantica a file condiviso immutabile

il file condivisibile non può essere modificato: il nome non viene riutilizzato e il contenuto non è modificabile

SEMANTICHE USATE per OPERAZIONI

Le semantiche più usate sono

Semantica di sessione (più immagini associate al file)

Più copie distinte associate alla sessione sul 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 unica di accesso al file

Esecuzione serializzata delle transazioni

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

Uso di lock che producono serie limitazioni al parallelismo

FILE SYSTEM DEPENDABILITY

Molte proprietà collegate alla garanzia del servizio o Dependability

Robustezza **reliability**

Un file è robusto, o in memoria stabile, se è garantita la sopravvivenza delle informazioni anche in caso di guasto dei dischi (senza garanzie sul tempo di recovery)

Ottenuto via 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 in correlazione ad altri file o utenti

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 e anche in tempi vincolati

DEPENDABILITY e REPLICAZIONE

Proprietà non equivalenti tra loro anche in conflitto

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

Un file reliable non è necessariamente recuperabile e viceversa

Tutte le proprietà si basano sulla ...

Replicazione dei file in diverse copie

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

requisito per replicazione

più copie di uno stesso file su macchine non correlate o failure-independent

la disponibilità di una replica non deve influire sulle altre

Gestione delle modifiche sulle diverse copie impone delle strategie di coordinamento: consistenza vs. disponibilità

USO dello STATO di INTERAZIONE

Servizi stateful e stateless

Un server può mantenere informazioni sulla connessione ed i servizi che sta offrendo ai cliente nello stato (stateful) o non mantenerlo affatto e lasciare il compito ai clienti (server stateless)

Server stateful

il server mantiene un identificatore univoco per ogni cliente e si permette anche un accesso a circuito virtuale tra cliente e server

Migliore performance, ma in caso di guasto, le informazioni del server sono perse e il recupero dello stato avviene con dialogo con i clienti o abort di tutte le operazioni in corso

Server stateless

il server non mantiene informazioni dei cliente ed ogni servizio è indipendente dal precedente e tipicamente idempotente

Maggiore robustezza ai guasti, ma richiesta alta ridondanza nelle operazioni e informazioni sulla rete

IMPLEMENTAZIONE di NOMI

Tecniche di ricerca dei file con trasparenza dell'allocazione

Necessità di **conversione del nome logico in nome di basso livello**

Necessità di limitare i costi di conversione e di usare la conversione più volte per dividerne il costo su tutti gli usi

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, con problemi di **costi non predicibili**

Identificatori strutturati

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

Un nome ha una parte che identifica la unità componente, cioè il server di allocazione del file, e una seconda che identifica il file sul server

IMPLEMENTAZIONE di NOMI

Hint (Consigli nell'Accesso)

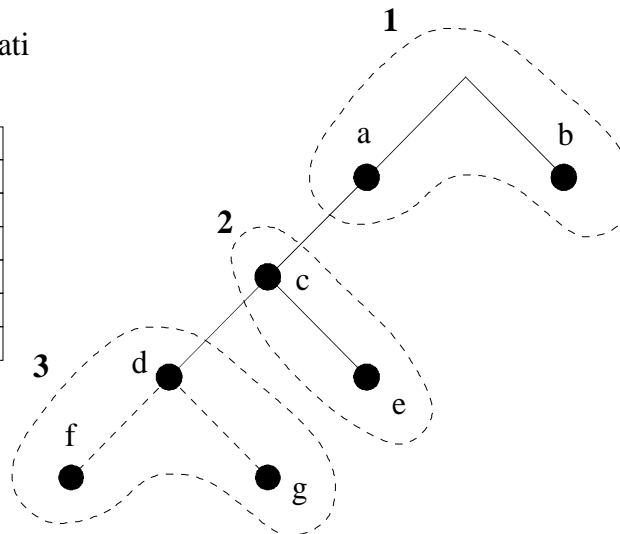
la ricerca di un file tramite hint (non sempre rigidamente validi),
cioè attraverso un semplice consiglio di ricerca

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

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

Tabella degli
Identificatori Strutturati

Unità	File
1	a
1	b
2	c
2	e
3	d
3	f
3	g



IMPLEMENTAZIONE di NOMI

Hint (Consigli nell'Accesso)

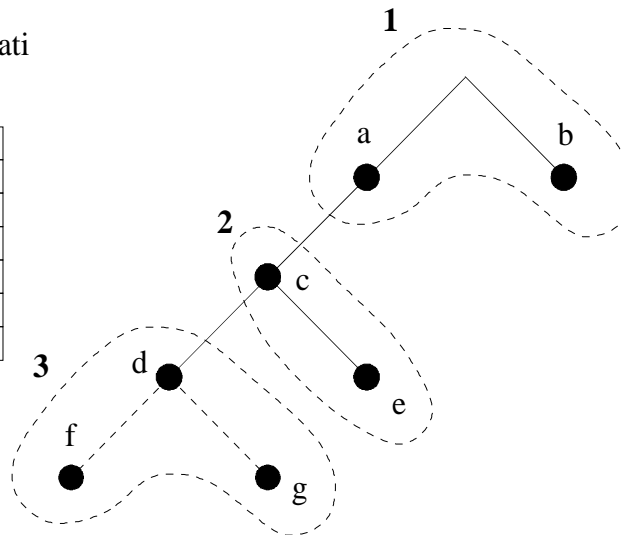
la ricerca di un file tramite hint (non sempre rigidamente validi),
cioè attraverso un semplice consiglio di ricerca

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

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

Tabella degli
Identificatori Strutturati

Unità	File
1	a
1	b
2	c
2	e
3	d
3	f
3	g



ACCESSO ai FILE REMOTI

Per il trasferimento dati due metodi complementari

- **servizio remoto (richiesta)** vs. **caching (mobilità e copia)**

Servizio remoto

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

corrispondenza tra accessi e traffico dal/al server

Caching

tecnica di caching (o buffering), memorizzazione temporanea in memoria locale al cliente

cache del cliente per ridurre il carico del server e il traffico generato

cache usato anche in combinazione con servizio remoto

In caso di cache, la copia principale è quella del server

problemi di consistenza cache: se copie nelle cache clienti, le modifiche di una cache devono essere propagata alla copia master e alle altre

CONVALIDA della CACHE

La copia in cache deve essere consistente con la principale: in caso di inconsistenza si deve eseguire un aggiornamento con un protocollo su iniziativa del cliente o del servitore

Client initiated approach

Il cliente controlla la consistenza delle informazioni sul server

La frequenza di controllo è importante: da un controllo per ogni accesso alla cache fino ad un solo controllo all'apertura del file, o ad intervallo

Controllo introduce ritardo nelle richieste di accesso

Server initiated approach (server con stato)

Il server controlla le copie di ciascun cliente che possiede cache

Se potenziale inconsistenza dei dati, il server 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

write through (scrittura e lettura immediata)

delayed write (operazioni differite e minore consistenza: anche write on close e periodic write)

Write through (e anche read through) Affidabilità ma scarsa efficienza
invio immediato dei dati modificati al server ed a tutti i client per aggiornare le cache (se noti) (sia lettura sia scrittura)

Delayed write (scrittura ritardata) Performance, problemi di consistenza
modifiche scritte nella cache locale e solo più tardi propagate al server e alle altre copie, letture dalla cache

Write on close problemi di consistenza ed affidabilità
scrittura alla chiusura, cioè aggiornamento solo alla chiusura del file
ottime performance in accessi prolungati con frequenti modifiche ad un file

Periodic write problemi di consistenza ed affidabilità
esame periodico della cache e aggiornamento solo dei blocchi modificati

IMPLEMENTAZIONI della CACHE

Dimensioni della cache

Granularità dei 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,
ma anche tempo richiesto per trasferire i dati e la probabilità di problemi di
inconsistenza

Allocazione della cache in memoria centrale o su disco

memoria veloce accesso ai dati anche per macchine diskless
disco, cresce tempo di accesso ai dati ma maggiori dimensioni ed affidabilità

Politiche di flushing

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

SISTEMI: UNIX United - (generazione 0)

I file system distribuiti cominciano a sperimentarsi nei primi anni 80 e hanno una diffusione limitata

La prima generazione è da intendersi come prototipi

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

Si propone un'unica organizzazione architetturale per l'accesso ai file con ogni sistema componente rappresentato da una directory che ha un nome non trasparente e viene richiesta in modo esplicito

Si può accedere ai file su ogni nodo visibile via il directory

- non trasparenza alla allocazione
- buon livello di autonomia ed espandibile
- nessuna replicazione delle copie ed affidabilità

UNIX United - (generazione 0)

Il sistema viene usato nel campus di NewCastle

Supporto non molto efficiente

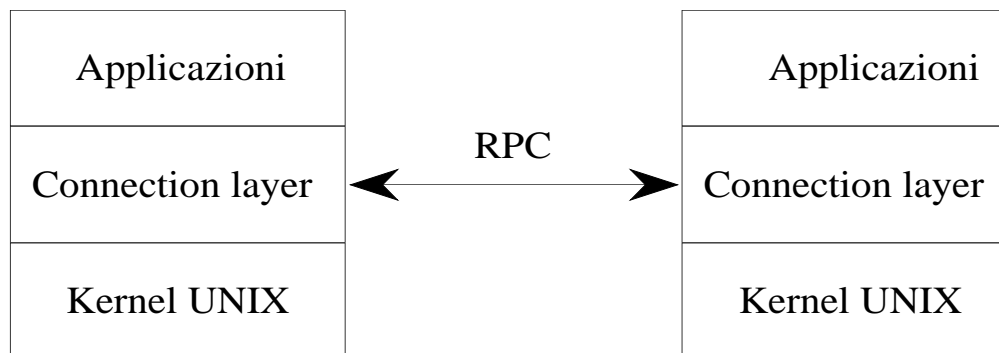
Uso di RPC per comunicare con Connection layer

Ricerca di un file avviene attraverso la esplorazione di tutte le macchine del percorso

Il sistema mantiene informazioni sull'indirizzo e il routing per accedere ad ogni file descriptor

Si usano RPC e non si prevede nessuna forma di caching

le connessioni per accesso remoto sono di tipo stateful



NFS - (I generazione)

Network File System progettato per un numero limitato di workstation in rete e integrato nel kernel SUN OS per efficienza con obiettivo anche di integrazione con altre realizzazioni

L'obiettivo di condivisione di file in maniera trasparente e anche massima efficienza: server di tipo statelesse semantica UNIX

Meccanismo del mount che permette la trasparenza alla allocazione del file (non indipendenza dall'allocazione dei file che sono fissi)

Naming a parziale condivisione cliente responsabile dei nomi:

il cliente non vede ciò che il server ha montato ma il file system locale al server
il cliente può montare su parti di file system che sono già derivati da server (cascade mount)

Virtual F.S. usa una struttura detta v-node

Una Mount table locale al cliente identifica macchina ed i-node per il file che viene acceduto tramite RPC

Si prevedono cache locali

NFS - IMPLEMENTAZIONE

Il v-node specifica di passare la richiesta ad un altro sistema

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

RPC (versione 3) consente operazioni di:

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

mancano operazioni di stato (tipo lock)

(cosa succede in caso di operazioni concorrenti?)

NFS usa due meccanismi di cache del cliente a tempo

Per i file, a memorizzazione locale di file (3 sec.)

Per file attribute, (direttori 30 sec.)

Uso di informazioni sui server e sugli i-node virtuali utilizzati recentemente

Locus File System - (II generazione)

LOCUS estende il kernel di UNIX per realizzare un file system globale con proprietà di tolleranza ai guasti: unico sistema di nomi

ogni macchina ha una vista completa e consistente del F.S. virtuale

La struttura dei file logici prevede sia replicazione sia trasparenza e indipendenza dall'allocazione gestite dal sistema

Unità componente dei file replicati, il **filegroup logico**, formato da un insieme di copie di file fisici **in astratto**, una struttura unica

in concreto, fisicamente mappato in un multiple file container memorizzato e replicato in maniera trasparente

Il file designator è la struttura dati di basso livello del file indipendente dalla allocazione **{numero di filegroup, numero di i-node}**

File system con nomi a completa condivisione

Il sistema mantiene una tabella per ogni filegroup con un file designator della directory logica corrente e il nodo per l'accesso sincronizzato al file

Locus File System: ACCESSO ai FILE

Accesso ai file in LOCUS coinvolge tre entità su nodi diversi

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

Si sceglie ss per le operazioni di apertura del file e si mantengono informazioni di consistenza delle repliche

Al primo accesso, css cerca tra ss la versione aggiornata e la passa a us

Operazioni di lettura accedono da us ad ss direttamente

In scrittura si media sempre attraverso css: **css seleziona una copia primaria, primary copy, in cui registra le modifiche con delayed write**

Totale trasparenza e indipendenza dall'allocazione

Tecniche di read-ahead e caching sia di dati sia di direttori

Connessione tipo stateful: ss e css mantengono informazioni di sessione

Locus File System: IMPLEMENTAZIONE

LOCUS utilizza il meccanismo di **shadow page** per la consistenza dei dati e la atomicità operazioni di scrittura

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 la sincronizzazione **UNIX** in distribuito con una politica di 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

Locus permette di lavorare anche in caso di partizione del sistema

SUPPORTO ad-hoc efficiente

Uso di funzioni di comunicazione integrate nel kernel

Il supporto realizzato con processi leggeri interni al kernel attivati per ogni richiesta di file

Sprite - (II generazione)

SPRITE sistema operativo distribuito sperimentale

basato su workstation con alta capacità di memoria e cache
Necessità di caricare e mantenere in memoria interi file system
(ipotesi 500Mbyte di memoria senza disco locale)

Unica struttura UNIX con trasparenza dell'allocazione e semantica di condivisione UNIX, mantenendo efficienza della implementazione

SPRITE kernel integra la 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

Sprite - IMPLEMENTAZIONE

Il client seleziona il server dalla propria tabella

se la richiesta fallisce, il client manda un messaggio broadcast con il nome del file ed il server con il file risponde con il proprio prefix
uso di hint rappresentato dalla tabella e di messaggi broadcast che limitano la scalabilità

SPRITE usa un metodo ibrido per convalida cache

ogni apertura in scrittura incrementa la versione del file: il client confronta la propria versione con il server, e se differenti, il client 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 si disabilita il caching e si deve mantenere un migliore aggiornamento delle copie

Andrew File System - (II generazione)

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

Ipotesi di architettura: distinzione di macchine client e server

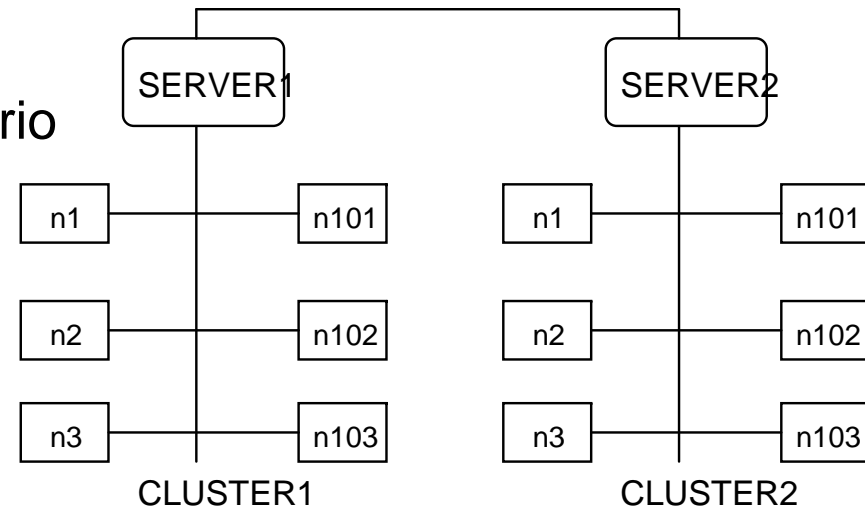
Client workstation con spazio per file sia locale sia condiviso

Server dedicati alla memorizzazione trasparente dello spazio condivisi

Per ottenere **scalabilità, cluster di workstation e server collegati tramite un server detto router al backbone**

In genere, si privilegia l'accesso a risorse locali: ogni cliente vede il proprio spazio locale e anche lo spazio del server proprio come condiviso

la località viene attuata
con il movimento di interi
file (o blocchi di file)



Andrew File System - IMPLEMENTAZIONE

Indipendenza dell'allocazione con nomi a completa condivisione

Spazio condiviso costituito da unità, o volumi, unite tra loro tramite un meccanismo simile al mount: le informazioni di allocazione sono replicate su ogni server cui accedono i clienti

I volumi possono migrare e 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 con interazione con il server solo alla apertura/chiusura: il cliente lavora in locale

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

Lo stato può produrre problemi nel ripristino in caso di guasti

ANDREW FS definito in termini di processi leggeri

server prevedono processi leggeri con nonpreemptive scheduling per soddisfare le richieste dei client

CODA - (evoluzione della II generazione)

CODA come evoluzione di AFS per operazioni disconnesse

uso estensivo di copie dei file, residenti in diverse località del sistema

- Vice l'insieme dei file server
- Virtue nodi utente in cui sono copiati i file all'accesso
- Venus processo utente che accede ai file

I file server possono replicare i file e le copie attraverso un indicatore di gruppo Volume Storage Group

I nodi utente possono vedere un solo sottoinsieme dei nodi server Vice dipendentemente dallo stato di rete e località

Un cliente in accesso a un file, riceve la copia del file e anche un meccanismo per la distribuzione dei file al/ai server (callback promise)

Le operazioni sulla copia locale in modo ottimista e possono continuare anche se il gruppo dei server noti VSG si svuota (operazioni disconnesse)

Al completamento delle operazioni su un file una copia del file viene mandata in broadcast a tutti i server del gruppo VSG con possibilità di inconsistenza

CODA - CONSISTENZA

Si noti che la capacità di lavorare in caso di partizione, permette anche di operare meglio in caso di clienti mobili o wireless

Ogni server CODA mantiene un vettore di interi per ogni file, che riporta un intero per stimare la versione del file per il server di indice corrispondente e anche indice delle versioni che il server mantiene

Se un server trova che le versioni sono diverse, deve riconciliare le copie caricando le versioni mancanti

In caso di conflitto di versioni, si ricorre a gestori di riconciliazione automatizzati che cercano di produrre consistenza altrimenti, si chiede un intervento manuale

Operazioni disconnesse su cache con politiche di carico del file

in stato disconnesso alcune operazioni non sono possibili ma una copia in cache consente di andare avanti e fare azioni anche di scrittura

si prepara un log ottimizzato in lunghezza e contenuto (cancellando cambiamenti sovrascritti) pronto per essere instaurato appena ci si riconnette con trattamento automatico della riconciliazione

Estensioni possibili (III generazione)

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

OceanStore sistema cooperativo di gestione di informazioni ad alta availability con accesso ubiquo e pervasivo da ovunque e garantita protezione delle informazioni, nel senso della autenticazione, cifratura, integrità

Il progetto si propone di fornire la capacità di memorizzare tutti i possibili file per i possibili utenti per un elevato numero di file:

10^{14} file per anno o anche 15 Exabytes / anno (10^{18})

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

I sistemi lavorano in modo indistinguibile

Si parla molto di Peer to Peer o anche P2P

OCEANSTORE: SISTEMA di NOMI

OceanStore con replicazione e mobilità dei file

il Sistema di nomi associa ad ogni file un GUID unico e permanente

Identificatore unico detto GUID globally unique identifier

Si organizza una gerarchia di nomi per le informazioni con i nomi mappati sui GUID via chiavi con Sistemi di hash (hash a 160 bit come chiave per ritrovare i file)

La ricerca dei file, parte da alcuni nodi root, per consentire un routing efficiente e robusto con informazioni replicate

La allocazione dei GUID ai diversi nodi root della struttura permette un primo livello di routing

Il secondo livello permette di trovare la vera allocazione dei file in uno scenario dinamico organizzato su molti server

Replicazione su molti server diversi in zone diverse dei file

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

OCEANSTORE: FILE SYSTEM

I componenti di OceanStore sono in fase di sviluppo prototipale e di ulteriore approfondimento

Molto interesse per la sua globalità, replicazione, e vastità di scope...

Oltre che capacità di adattamento alle richieste

Controllo degli Accessi sugli oggetti

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

Forma di Archivio

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

Sistemi P2P e non Cliente/Servitore

I nuovi sistemi si basano su **Peer-to-Peer (P2P)** (III generazione)

I sistemi P2P sono caratterizzati da nodi tutti allo stesso livello, dal punto di vista di responsabilità e protocolli, quindi si comportano sia da clienti e da servitori

Obiettivo applicativo: **alta disponibilità e alto grado di replicazione**

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

Realizzazione di file system come copie di informazioni consentendo anche i gradi di sicurezza più adatti alle specifiche

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

I sistemi più diffusi sono serviti per la distribuzione di informazioni speciali file musicali da distribuire ai clienti

Sistemi P2P ... solo i primi sistemi

Napster, gnutella, ... sistemi molto noti per condivisione di informazioni musicali o multimediali

molto diffusi come sistemi ad architettura con gradi di distribuzione crescente

Freenet file system (come OceanStore)

elevato grado di replicazione ed a diffusione derivante dalle richieste
strategie di propagazione ispirate al routing globale

Freehaven

informazioni più suddivise per robustezza e con alto grado di sicurezza

Publius

condivisione di informazioni con diversi gradi di sicurezza e privacy

SETI@home o SETI - applicazione di ricerca ed analisi di segnali extraterrestri eseguita su macchine clienti

Jabber - supporto alla conversazione e cooperazione tra utenti