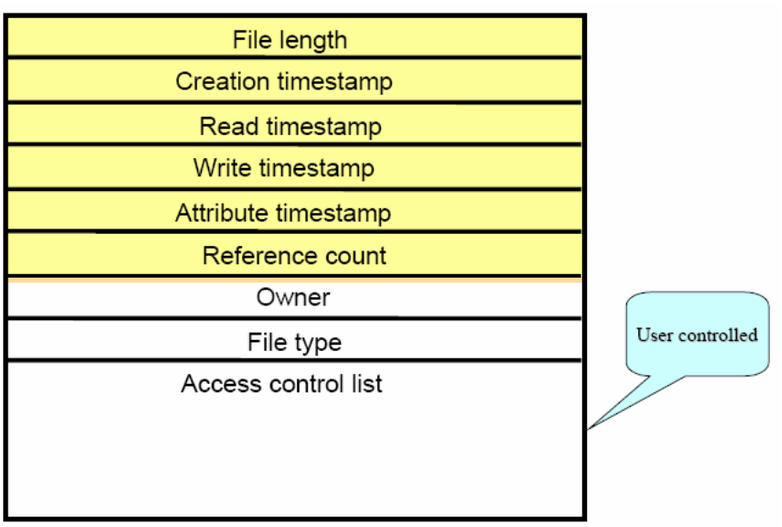
Sistem File Terdistribusi

Layanan file terdistribusi
• Layanan Dasar
  – tempat penyimpanan tetap untuk data dan program
  – operasi terhadap file (create, open, read,…)
  – multiple remote clients (dalam intranet)
  – file sharing
  – menggunakan semantic one-copy update umum, melalui RPC
• Perkembangan baru
  – persistent object stores (storage of objects)
• Persistent Java, Corba, …
– replikasi, caching keseluruhan file
–multimedia terdistribusi (contoh: file server Tiger video)

Operasi terhadap files (=data + attributes)
– create/delete
– attribute query/modify
– open/close
– read/write
– access control

• Organisasi Storage (tempat penyimpanan)
  – directory structure (hierarchical, pathnames)
  – metadata (file management information)
• file attributes
• informasi struktur directory, etc

File attributes



Struktur bertingkat sistem file

Directory module:	relates file names to file IDs
File module:	relates file IDs to particular files
Access control module:	checks permission for operation requested
File access module:	reads or writes file data or attributes
Block module:	accesses and allocates disk blocks
Device module:	disk I/O and buffering

Konsentrasipada tingkat yang lebih tinggi.

Kebutuhan sistem File Terdistribusi

- Transparency (clients tidak perlu tahu terhadap bentuk/mekanisme distribusi)
  - access transparency (client tidak perlu tahu bahwa file adalah terdistribusi, karena memiliki interface yang sama untuk file local/remote)
  - location transparency (name space file yang uniform terhadap client workstation)
  - mobility transparency (files dapat dipindahkan dari satu server ke lainnya tanpa berdampak pada client)
  - performance transparency (client performance tidak berdampak pada load layanan)
  - scaling transparency (kemungkinan perluasan apabila jumlah client bertambah)

- Concurrent file updates (perubahan yang dilakukan oleh satu client tidak berdampak pada yang lainnya)
- File replication (untuk load sharing, fault-tolerance)
- Heterogeneity (interface platform-independent)
- Fault-tolerance (tetap beroperasi secara kontinu meskipun terjadi kesalahan pada client ataupun server)
- Consistency (*one-copy-update* semantics atau slight variations)
- Security (access control)
- Efficiency (unjuk kerja yang comparable terhadap sistem file conventional)

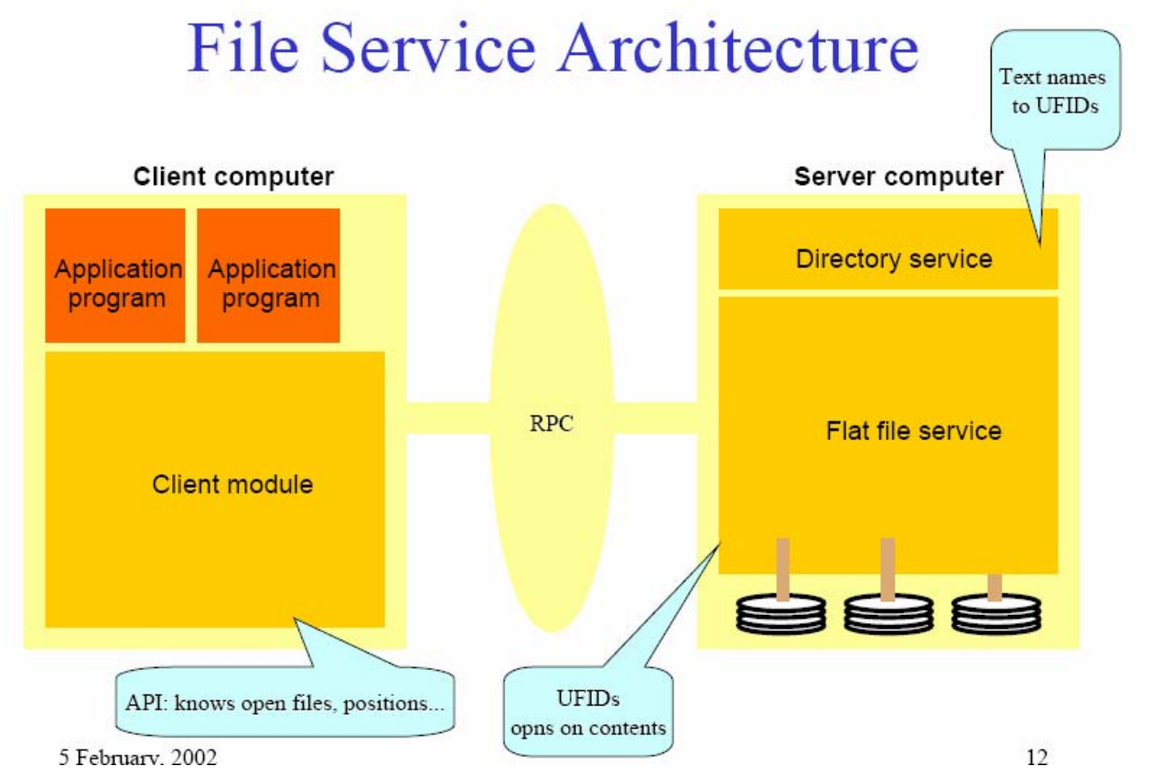
Opsi Perancangan Layanan File

• Stateful

– server menyimpan informasi tentang file yang open, posisi sekarang (current position) dan file locks
– open (dibuka) sebelum access dan kemudian ditutup
– performa yang lebih baik – pesan yang lebih pendek, dimungkinkan untuk read-ahead
– server failure - kehilangan state
– client failure - tables fill up
– menyediakan file locks

Stateless

– server tidak menyimpan state informasi
– file operations idempotent, harus mengandung semua yang diperlukan (longer message)
– perancangan file server yang lebih simpel
– dapat dengan mudah di-recovery apabila client ataupun server crash
– locking membutuhkan extra lock server untuk mempertahankan state



Arsitektur File server
Components (untuk openness):
• Layanan Flat file
– operations pada file contents
– unique file identifiers (UFIDs)
– penterjemahan dari UFIDs ke lokasi file
• Layanan Directory
– pemetaan antara nama sbg text ke UFIDs
• Modul Client
– API untuk file access, satu untuk setiap computer client
– menyimpan status (state): open files, positions
– mengetahui lokasi jaringan dari flat file dan directory server

Layanan Flat file melalui RPC interface
• Digunakan oleh modul client, bukan user programs
– *FileId* (UFID) mendefinisikan file secara tunggal
– mengirim pesan invalid jika file tidak ada atau akses yang tidak wajar.
– *Read/Write*; *Create/Delete*; *Get/SetAttributes*
• No open/close! (unlike UNIX)
– akses secara langsung dan segera dengan *FileId*
– *Read/Write* mengidentifikasi saat dimulai
• Meningkatkan fault-tolerance
– operations idempotent kecuali Create, dapat diulang (semantic atleast-once RPC)
– stateless service

Flat file service operations

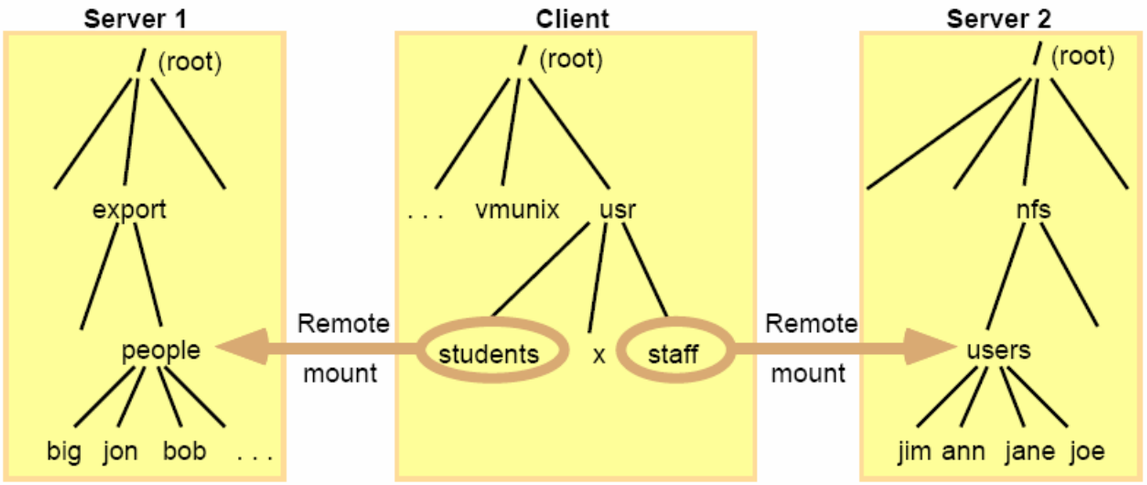
Read(FileId, i, n) -> Data — throws *BadPosition*	If $1 \le i \le Length(File)$: Reads a sequence of up to n items from a file starting at item i and returns it in *Data*.
Write(FileId, i, Data) — throws *BadPosition*	If $1 \le i \le Length(File)+1$: Writes a sequence of *Data* to a file, starting at item i, extending the file if necessary.
Create() -> FileId	Creates a new file of length 0 and delivers a UFID for it.
Delete(FileId)	Removes the file from the file store.
GetAttributes(FileId) -> Attr	Returns the file attributes for the file.
SetAttributes(FileId, Attr)	Sets the file attributes (only owner, file type and access control list; rest maintained by flat file service).

Access control
• Dalam sistem file UNIX
  – hak akses diperiksa berdasarkan mode akses (read,write, execute) ketika open
  – identitas user diperikasi ketika login,
• Dalam sistem terdistribusi
  – hak akses harus diperiksa pada server
• RPC unprotected
• dimungkinkan melakukan forging identity, tetapi merupakan risiko keamanan
  – user id biasanya dikirim bersamaan dengan setiap request (e.g. Sun NFS)
  – stateless

Struktur Directory
• Hierarchical
– tree-like, pathnames dari root
– (di UNIX) dapat digunakan beberapa nama untuk setiap file (*link* operation)
• Sistem penamaan (Naming system)
– diimplementasi oleh modul client, menggunakan directory service
– root memiliki well-known UFID
– melokasikan file mengikuti path dari root

File names
Text name (=directory pathname+file name)
• hostname: local name
– not mobility transparent
• struktur nama uniform (name space yang sama untuk semua clients)
• remote mount (e.g. Sun NFS)
– remote directory dimasukkan ke dalam local directory
– relies on clients maintaining consistent naming conventions across all clients
• all clients harus implement local tree yang sama
• harus melakukan mount remote directory ke local directory yang sama

Remote mount



Note: The file system mounted at */usr/students* in the client is actually the sub-tree located at */export/people* in Server 1; the file system mounted at */usr/staff* in the client is actually the sub-tree located at */nfs/users* in Server 2.

Layanan Directory
• Directory
– conventional file (client terhadap layanan flat file)
– memetakan text names ke UFIDs
• Operasi
– memerlukan *FileId*, machine readable UFID sebagai parameter
– locate file (*LookUp*)
– add/delete file (*AddName/UnName*)
– mencocokkan nama file terhadap ekspresi regular (*GetNames*)

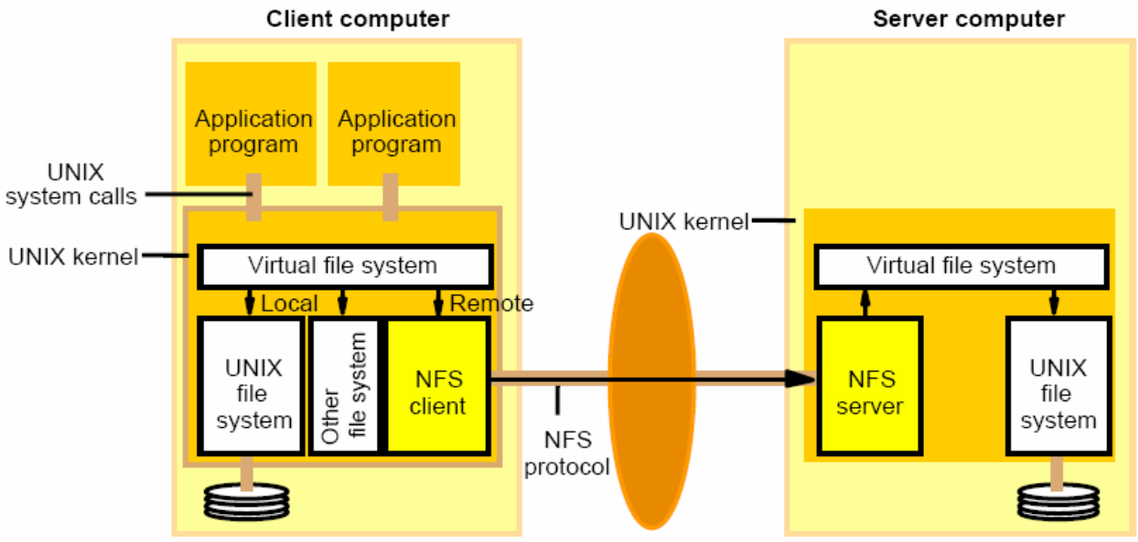
Directory service operations

Lookup(Dir, Name) -> FileId — throws *NotFound*	Locates the text name in the directory and returns the relevant UFID. If *Name* is not in the directory, throws an exception.
AddName(Dir, Name, File) — throws *NameDuplicate*	If *Name* is not in the directory, adds (*Name*, *File*) to the directory and updates the file's attribute record. If *Name* is already in the directory: throws an exception.
UnName(Dir, Name) — throws *NotFound*	If *Name* is in the directory: the entry containing *Name* is removed from the directory. If *Name* is not in the directory: throws an exception.
GetNames(Dir, Pattern) -> NameSeq	Returns all the text names in the directory that match the regular expression *Pattern*.

File sharing
Multiple clients berbagi (share) file yang sama untuk akses read/write.
• One-copy update semantics
– setiap read melihat dampak semua writes sebelumnya

– suatu proses write akan segera visible ke client yang sudah siap membuka file untuk reading
• Problems!
– caching: memelihara konsistensi antara beberapa copies yang sulit di achieve
– akses serialisasi dengan menggunakan file locks (memperngaruhi performance)
– trade-off antara consistency dan performance

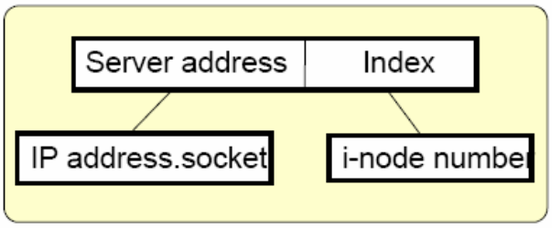
NFS architecture



File identifier (*FileId*)

Simple Solution

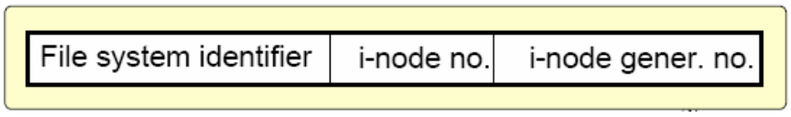
– i-node (number identifying file within file system)
– file migration requires finding and changing all *FileId*s
– UNIX reuses i-node numbers after file deleted (i-node gen. no)



NFS file handle

Virtual file system uses i-node if local, file handle if remote.

File handle



Caching dalam NFS

• Indispensable untuk performance
• Caching
– menyimpan data yang baru saja dipakai (file pages, directories, file attributes) dalam cache
– updates data in cache untuk kecepatan
– ukuran block biasanya 8kbytes
• Server caching
– cache dalam server memory (UNIX kernel)
• Client caching
– cache dalam client memory, local disk

Server caching
• Menyimpan data dalam server memory
• Read-ahead: anticipate pages mana yang dibaca
• Delayed write
– update dalam cache; menulis ke disk secara periodic (UNIX *sync* ke synchronise cache) atau ketika membutuhkan tempat.
– which contents seen by users depends on timing
• Write through
– cache dan menulis ke disk (reliable, poor performance)
• Write pada saat menutup
– menulis ke disk hanya ketika menerima commit (cepat tetapi bermasalah dengan files open untuk waktu yang lama).

Client caching
• Berpotensi terhadap masalah konsistensi!
– menmbulkan berbagai versi, bagian file, … karena proses menulis yang di-delay
– clients melakukan polling server untuk memeriksa apakah proses copy masih valid
• Timestamp method
– tag untuk chek validasi terakhir dan waktu modifikasi terakhir,
– copy valid jike waktu secak check terkahir kurang dari freshness interval, atau waktu modifikasi pada modification time pada server yang sama.
– choose freshness interval secara adaptive, 3-30 sec untuk files, 30-60 sec untuk directories
– untuk small freshness interval, berpotensi heavy load pada network
• Reads
– perform validity check ketika cache entry digunakan
– jika tidak valid, request data dari server
– several optimisations untuk mengurangi traffic
– recent updates tidak sealalu visible (timing!)
• Writes
– when page dimodifikasi, diberi tanda kotor (dirty)
– dirty pages flushed asynchronously, periodically (client's synch) dan pada saat close
• Not truly *one-copy update* semantics...

New developments
• **AFS**, Andrew file system (CMU 1986)
– whole-file serving (64kbytes), whole-file caching (pada local client disk, ratusan file yang baru digunakan)
• **NFS** protocol (precise one-copy update semantics)
– Spritely NFS: memperluas open/close dengan state info yang disimpan di server, menambah server callbacks untuk memberitahu tentang cache entries
• **WebNFS** (Program Internet dapat berinteraksi secara langsung ke NFS server, bypassing mount).
• **xFS** (serverless network, file serving bertanggung jawab terhadap distribusi diantara LAN)