PAYAP UNIVERSITY

Distributed File Systems

Vaidė Narváez

Computer Information Systems

July 21st, 2010

© 2010 Vaidė Narváez

PAYAP UNIVERSITY

Storage systems and their properties

	Sharing	*Persis-tence*	*Distributed cache/replicas*	*Consistency maintenance*	*Example*
Main memory	✗	✗	✗	1	RAM
File system	✗	✓	✗	1	UNIX file system
Distributed file system	✓	✓	✓	✓	Sun NFS
Web	✓	✓	✓	✗	Web server
Distributed shared memory	✓	✗	✓	✓	Ivy (DSM, Ch. 18)
Remote objects (RMI/ORB)	✓	✗	✗	1	CORBA
Persistent object store	✓	✓	✗	1	CORBA Persistent Object Service
Peer-to-peer storage system	✓	✓	✓	2	OceanStore (Ch. 10)

Types of consistency:
1: strict one-copy. ✓: slightly weaker guarantees. 2: considerably weaker guarantees.

©Addison-Wesley 2005

© 2010 Vaidė Narváez

PAYAP UNIVERSITY

Characteristics of file systems

- Organization
- Storage
- Retrieval
- Naming
- Sharing
- Protection of files

© 2010 Vaidė Narváez

Characteristics of file systems

- Data and attributes

© 2010 Vaidė Narváez

PAYAP UNIVERSITY

Characteristics of file systems

- Data and attributes

File length
Creation timestamp
Read timestamp
Write timestamp
Attribute timestamp
Reference count
Owner
File type
Access control list

©Addison-Wesley 2005

PAYAP UNIVERSITY

Characteristics of file systems

- Data and attributes

File length
Creation timestamp
Read timestamp
Write timestamp
Attribute timestamp
Reference count
Owner
File type
Access control list

©Addison-Wesley 2005

- A directory is a file, often of a special type, that provides mapping from text names to internal file identifiers

Layered module structure of FS

Module	Function
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

ⓒAddison-Wesley 2005

ⓒ 2010 Vaidė Narváez

PAYAP UNIVERSITY

File system operations

filedes = open(name, mode) *filedes = creat(name, mode)*	Opens an existing file with the given *name*. Creates a new file with the given *name*. Both operations deliver a file descriptor referencing the open file. The *mode* is *read*, *write* or both.
status = close(filedes)	Closes the open file *filedes*.
count = read(filedes, buffer, n) *count = write(filedes, buffer, n)*	Transfers *n* bytes from the file referenced by *filedes* to *buffer*. Transfers *n* bytes to the file referenced by *filedes* from buffer. Both operations deliver the number of bytes actually transferred and advance the read-write pointer.
pos = lseek(filedes, offset, whence)	Moves the read-write pointer to offset (relative or absolute, depending on *whence*).
status = unlink(name)	Removes the file *name* from the directory structure. If the file has no other names, it is deleted.
status = link(name1, name2)	Adds a new name (*name2*) for a file (*name1*).
status = stat(name, buffer)	Gets the file attributes for file *name* into *buffer*.

©Addison-Wesley 2005

© 2010 Vaidė Narváez

PAYAP UNIVERSITY

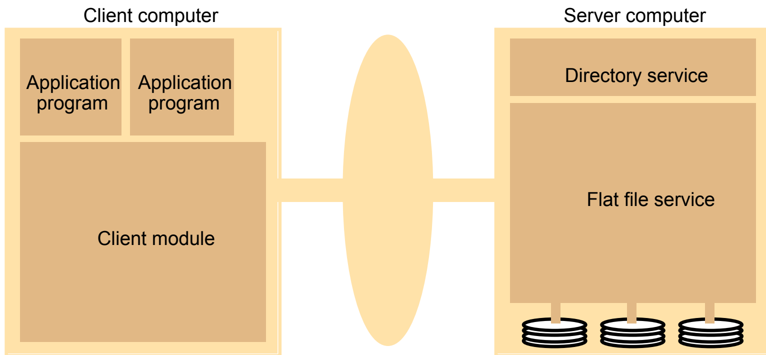
Distributed FS requirements

- transparency:
  - access
  - location
  - mobility
  - performance
  - scaling
- concurrent file updates
- file replication
- consistency
- fault tolerance
- hardware and os heterogeneity
- security
- efficiency

© 2010 Vaidė Narváez

PAYAP UNIVERSITY

File service architecture



©Addison-Wesley 2005

© 2010 Vaidė Narváez

File service architecture

- Flat file service
  - unique file identifiers (UFID)
- Directory service
  - map names to UFIDs
- Client module
  - integrate/extend flat file and directory services
  - provide a common application programming interface (can emulate different file interfaces)
  - stores location of flat file and directory services

Flat file service interface

- RPC used by client modules
  - not by user-level programs (which use client modules)
- More fault tolerant compared to UNIX
  - Repeatable (idempotent) operations
    - at-least-once semantics
    - no open (hence close) so no state to remember
    - specify starting location and UFID (from directory service) in Read/Write
  - Stateless server
    - Can be restarted without the server or client restoring any state information

© 2010 Vaidė Narváez

Flat file service operations

Read(FileId, i, n) -> Data — throws *BadPosition*	If $1 \leq i \leq Length(File)$: 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 $1 \leq i \leq Length(File)+1$: 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 those attributes that are not shaded in Figure 8.3).

©Addison-Wesley 2005

© 2010 Vaidė Narváez

Access control

- UNIX checks access rights when a file is opened
  - subsequent checks during read/write are not necessary
- distributed environment
  - access rights are checked at the server
  - stateless approaches
    1. access check whenever UFID is issued
       - client gets an encoded ”capability” (who can access and how)
       - capability is submitted with each subsequent request
    2. access check for each request.

© 2010 Vaidė Narváez

PAYAP UNIVERSITY

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, FileId) — 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*.

©Addison-Wesley 2005

© 2010 Vaidė Narváez

PAYAP UNIVERSITY

Hierarchic File System

- Directories are arranged in a tree structure
- Any file or directory is reference by a *pathname*
- Files can have more than one name (*link*)

© 2010 Vaidė Narváez

File Groups

A file group is a logical collection of files located on a given server

- a server can have more than one group
- a group can change server
- *filesystems* in unix

© 2010 Vaidė Narváez

Sun NFS

- Industry standard for local networks since the 1985
- OS independent
- unix implementation
- rpc
- udp or tcp

NFS Illustrated

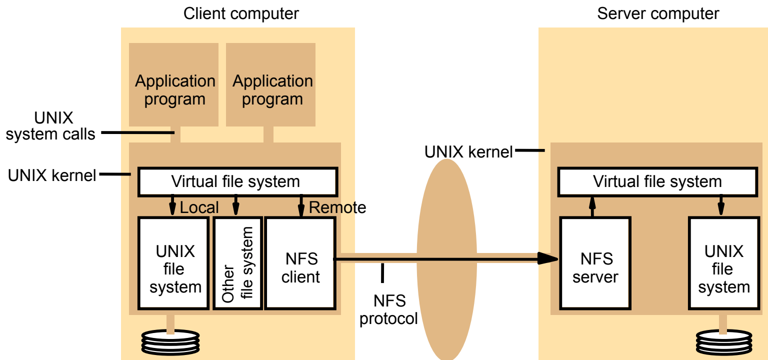
Brent Callaghan

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

© 2010 Vaidė Narváez

PAYAP UNIVERSITY

Architecture



©Addison-Wesley 2005

© 2010 Vaidė Narváez

Virtual File System

- access transparency
- part of unix kernel
- NFS *file handle*, 3 components:
  - filesystem identifier
    - different groups of files
  - i-node (index node)
    - structure for finding the file
  - i-node generation number
    - i-nodes are reused
    - incremented when reused

© 2010 Vaidė Narváez

PAYAP UNIVERSITY

NFS operations

lookup(dirfh, name) -> fh, attr	Returns file handle and attributes for the file *name* in the directory *dirfh*.
create(dirfh, name, attr) -> newfh, attr	Creates a new file name in directory *dirfh* with attributes *attr* and returns the new file handle and attributes.
remove(dirfh, name) status	Removes file name from directory *dirfh*.
getattr(fh) -> attr	Returns file attributes of file *fh*. (Similar to the UNIX *stat* system call.)
setattr(fh, attr) -> attr	Sets the attributes (mode, user id, group id, size, access time and modify time of a file). Setting the size to 0 truncates the file.
read(fh, offset, count) -> attr, data	Returns up to *count* bytes of data from a file starting at *offset*. Also returns the latest attributes of the file.
write(fh, offset, count, data) -> attr	Writes *count* bytes of data to a file starting at *offset*. Returns the attributes of the file after the write has taken place.
rename(dirfh, name, todirfh, toname) -> status	Changes the name of file *name* in directory *dirfh* to *toname* in directory to *todirfh*
link(newdirfh, newname, dirfh, name) -> status	Creates an entry *newname* in the directory *newdirfh* which refers to file *name* in the directory *dirfh*.

©Addison-Wesley 2005

© 2010 Vaidė Narváez

NFS operations

symlink(newdirfh, newname, string) -> status	Creates an entry *newname* in the directory *newdirfh* of type symbolic link with the value *string*. The server does not interpret the *string* but makes a symbolic link file to hold it.
readlink(fh) -> string	Returns the string that is associated with the symbolic link file identified by *fh*.
mkdir(dirfh, name, attr) -> newfh, attr	Creates a new directory *name* with attributes *attr* and returns the new file handle and attributes.
rmdir(dirfh, name) -> status	Removes the empty directory *name* from the parent directory *dirfh*. Fails if the directory is not empty.
readdir(dirfh, cookie, count) -> entries	Returns up to *count* bytes of directory entries from the directory *dirfh*. Each entry contains a file name, a file handle, and an opaque pointer to the next directory entry, called a *cookie*. The *cookie* is used in subsequent *readdir* calls to start reading from the following entry. If the value of *cookie* is 0, reads from the first entry in the directory.
statfs(fh) -> fsstats	Returns file system information (such as block size, number of free blocks and so on) for the file system containing a file *fh*.

©Addison-Wesley 2005

Client Integration

- NFS client emulates Unix file semantics

- in the kernel, not in a library, because:
  - access files via system calls
  - single client module for multiple user processes
  - encryption can be done in the kernel

© 2010 Vaidė Narváez

Access control

- NFS server is stateless, doesn’t keep open files for clients
- server checks identity each time (uid and gid)

© 2010 Vaidė Narváez

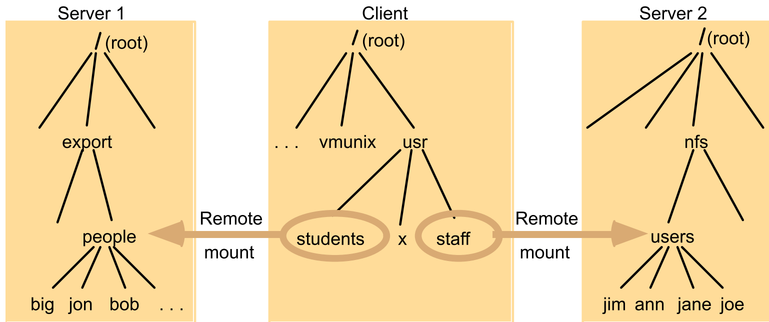
Mount service

- the process of including a new filesystem is called mounting
- `/etc/exports` has filesystems that can be mounted by others
- clients use a modified mount command for remote filesystems
- communicates with the mount process on the server in a mount protocol
- hard-mounted
  - user process is suspended until request is successful
  - when server is not responding
  - request is retried until it's satisfied
- soft-mounted
  - if server fails, client returns failure after a small # of retries
  - user process handles the failure

© 2010 Vaidė Narváez

PAYAP UNIVERSITY

Local and Remote FS



©Addison-Wesley 2005

© 2010 Vaidė Narváez

PAYAP UNIVERSITY

Pathname translation

- pathname: `/users/students/dc/abc`
- server doesn't receive the entire pathname for translation why?
- client breaks down the pathnames into parts
- iteratively translate each part
- translation is cached

© 2010 Vaidė Narváez

Automounter

- what if a user process references a file on a remote filesystem that is not mounted?

PAYAP UNIVERSITY

Automounter

- what if a user process references a file on a remote filesystem that is not mounted?
- table of mount points (pathname) and servers
- NFS client sends the reference to the automounter
- automounter finds the first server that is up
- mounts it at some location and sets a symbolic link (original impl)
- mounts it at the mount point (later impl)
- could help for fault tolerance, the same mount point with multiple replicated servers.

© 2010 Vaidė Narváez

Server Caching

- caching file pages, directory and file attributes
- read-ahead: prefetch pages following the most-recently read file pages
- delayed-write: write to disk when the page in memory is needed for other purposes
- ”sync” flushes ”dirty” pages to disk every 30 seconds
- two write options:
  - write-through: write to disk before replying to the client
  - cache and commit: - stored in memory cache
    - write to disk before replying to a ”commit” request from the client

© 2010 Vaidė Narváez

PAYAP UNIVERSITY

Client Caching

- caches results of `read, write, getattr, lookup, readdir`
- clients responsibility to poll the server for consistency
- Reading:
  - timestamp-based consistency validation
    -T_c: time when the cache entry was last validated
    -T_m: time when the block was last modified at the server
    - cache entry is valid if: $(T - T_c < t) \vee (T_m client = T_m server)$, where t is the freshness interval
  - validation doesn't guarantee the same level of consistency as one-copy

© 2010 Vaidė Narváez

Client Caching: writing

- dirty: modified page in cache
- flush to disk: file is closed or sync from client
- bio-daemon (block input-output)
  - read-ahead: after each read request, request the next file block from the server as well
  - delayed write: after a block is filled, it's sent to the server
  - reduce the time to wait for read or write

© 2010 Vaidė Narváez

Performance

- overhead is low
- main problems:
  - frequent `getattr()` for cache validation (piggybacking)
  - relatively poor performance if write-through is used on the server (delay-write or commit in current versions)
- write $<$ 5%
- lookup is almost 50% (step by step pathname translation)

© 2010 Vaidė Narváez

Summary NFS

- access transparency: same system calls for local or remote files
- location transparency: could have a single name space for all files (depending on all the clients to agree the same name space)
- mobility transparency: mount table need to be updated on each client (not transparent)
- scalability: can usually support large loads, add processors, disks, servers...
- file replication: read-only replication, no support for replication of files with updates
- hardware and OS: many ports
- fault tolerance: stateless and idempotent
- consistency: not quite one-copy for efficiency
- security: added encryption–Kerberos
- efficiency: pretty efficient, wide-spread use

© 2010 Vaidė Narváez

Other DFS

Google FS, Hadoop, Amazon S3...

© 2010 Vaidė Narváez