CSCE 455/855
Distributed Operating Systems

Distributed File Systems

Steve Goddard
goddard@cse.unl.edu

http://www.cse.unl.edu/~goddard/Courses/CSCE855

1

Overview

- File system is a key component of any distributed system
  - » Sometimes it is only used locally
  - » Many computations are most conveniently described when using files as shared resources available to all components of the distributed computation
  - » Distributed file systems are among the most well developed distributed system components because they are popular for support of pools of workstations
- File Service
  - » Specification of what the file system offers its clients
- File Server
  - » Process implementing the file service on a machine

2

Distributed File System Design

- Ideally a distributed file system should be *transparent*
  - » Computations and humans using it should not be able to tell that it is distributed
  - » This depends on transparency of several components
- Two major components
  - » File Service Interface
    - ❖ Operations on an individual file
  - » Directory Service Interface
    - ❖ Operations on groups of files
    - ❖ Name Space issues

3

File Service Interface

- File service interface answers fundamental questions
  - » What is a file?
  - » What can I do with and to a file?
- Files can vary in structure
  - » Sequences of records
  - » Complex record structures
  - » Sequence of bytes
- Sequence of bytes is most general, since the others can be implemented on top of it quite easily
  - » UNIX and Microsoft are also the most common distributed file systems and they both use this model
  - » Possible performance hit

4

File Service Interface

- Attributes
  - » Information associated with, but not part of, a file
    - ❖ Owner, size, creation time, access permissions
- Mutable/Immutable
  - » Can a file be modified after creation?
  - » We are used to this, but it makes distribution harder
  - » Immutable files only support CREATE and READ
  - » Simplifies caching and replication because it eliminates all consistency considerations
- Capabilities: one method of access control
  - » Objects explicitly granting access to holder
  - » May be passed from user to user

5

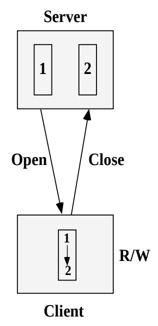
File Service Interface

- Access Control Lists
  - » Information associated with the file rather than the user
  - » Explicitly lists what users may access the file and what type of access is permitted to each
    - ❖ UNIX scheme
- Access Models
  - » Upload/Download
  - » Remote Access
  - » Trade simplicity against network traffic and latency
  - » Figure 5-1, page 247 Tanenbaum

6

File Service Interface
Upload/Download

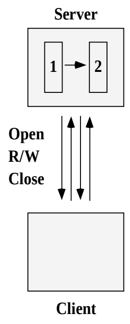
- Transfer file to client on OPEN or first READ
- Transfer back to server on CLOSE (last WRITE)
- R/W operations local on the client
- Conceptually simple
- Requires *lots* of client storage
- Large latencies associated with moving whole files
- Traffic could increase or decrease
  - » Depends on volume of R/W vs. file size



7

File Service Interface
Remote Access

- Support file operations remotely (Open/Close, R/W, etc)
- File remains on server
- Lowers startup latency but every operation goes across the network
  - » Network traffic could be larger or smaller than upload/download depending on volume of file operations vs. file size
- Requires less space on the client



8

Directory Service Interface

- Supports the file system structure
  - » Could be anything but virtually all systems use a hierarchical structure
    - ❖ Directed acyclic graph
    - ❖ Parent /Child relations and associated links
- Distribution (as usual) has all the usual problems, makes some of the normal ones worse, and has some new ones too
  - » Unique resolution of element in the name space to a file
  - » Composition of physical file systems (mount)
  - » Transparency - can users tell parts of FS are distributed
  - » Uniformity - is the name space the same on all machines

9

Directory Service Interface

- Key Issues:
  - » *Can* all machines have the same view of the FS?
  - » *Should* all machines have the same view of the FS?
  - » Performance considerations may make a common view undesirable even if it is possible
- Standard Implementation Strategy:
  - » Optimize most common case(s)
- Limit overhead by not distributing full FS view to all users
  - » Decreases distribution work while increasing management overhead required to decide who sees what
- Increase labor for less common operations (delete vs. read) by having the deleting system initiate analysis

10

Directory Service Interface
Name Resolution

- Name space can be arbitrary
  - » Hierarchy with names and slashes (forward or back) is the most common and seems to be the best
    - Uniformity of notation for all objects in FS
    - Maximum parsimony (succinct expression)
- Users and programs operate in the name space
  - » Path Name (name space element)
- Operating system uses its own internal designation
  - » Data structure reference (I-Node)
- Path Name to I-Node translation
  - » Provides access to all elements of the name space

11

Directory Service Interface
Name Resolution

- Universal use of the name space to represent all elements
  - » Requires Path→ I-Node translation to be smart about all types of objects
  - » Requires use of I-Node to represent all types of objects
    - Actual method used
- Name space object types
  - » File: most common elements
  - » Directory: also common elements
  - » Device Special File: access to device drivers
  - » Mount points: identifies physical file system borders
  - » Symbolic Links: useful and comparatively recent

12

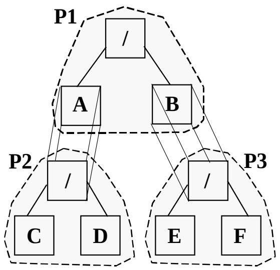
Directory Service Interface
File System Composition

- Name space is virtual but the FS contents are physical
  - » Must deal with multiple physical components
- Composition of multiple physical elements advantageous
  - » Graceful FS Scaling
    - ❖ Add arbitrary number of partitions to name space
  - » Location Transparency
    - ❖ Failure or replacement of physical partitions concealed
  - » Graceful Distribution
    - ❖ Distributed components distinguished within file system
    - ❖ Distributed element is just another physical partition
- Composition Operation: mount a partition on a directory

13

Directory Service Interface
File System Composition

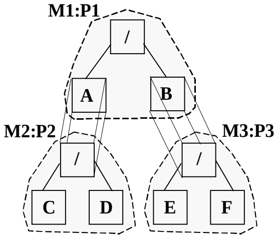
- Partition P1 is the root partition and provides the root (/) for FS
- Partitions P2 and P3 are separate physical partitions mounted on directories in P1
  - » A and B are P1 directories
  - » Covered by *mount* operation
    - ❖ mount /dev/P2 /A
- Each partition is a separate FS
  - » Separate I-Node pool
- Path→ I-Node: /A/C
  - » P1:/A marked as mount point redirects translation to P2:/



14

Directory Service Interface
File System Composition

- Consider partition Ids that include the machine identifier → M1:P1
- Path→ I-Node: /A/C
  - » P1:/A marked as mount point redirects translation to M2:P2:/
- Advantage here is that knowledge of distribution is limited to specific parts of the system
  - » Path→ I-Node
  - » I-Node operations
- I-Node ID must now include the machine: M1:I-Node
  - » Local Mx → local operation
  - » Remote Mx → remote operation



15

Directory Service Interface
File System Composition

- Composition (mounting) is also used to create a generic interface to a variety of file systems
  - » FTP based remote access
  - » WWW (HTTP) file systems
  - » Encrypted and compressed file systems
- Generalization of “file system” concept
  - » Generic file system support in many OS’s
  - » File system switch in Linux and others
- I-Node includes information on FS type
  - » Distribution easily supported as a file system type (NFS)
  - » File system data structure contains machine ID
    - ❖ I-Node structure need not change

16

Directory Service Interface
Transparency

- Location Transparency
  - » Path name gives no hint where the file is located
  - » Moving files physically can require many changes
    - Symbolic Links
- Location Independence
  - » Files can be moved without the path names changing
- /net/server/root is *not* location transparent but is location independent if *server* remains constant over moves
- Many (most?) installations using NFS still tend toward path names including a server (machine) component
  - » Many that are more transparent use symbolic links whose mappings change when things move

17

Directory Service Interface
Uniformity

- Can the file system name space look the same at every machine?
- Should the file system name space look the same at every machine?
- Uniformity implies that every machine have access to every file
- Transitive property of sharing
  - » Any two hosts sharing a file A implies that all hosts can see A
  - » Else the name space for the hosts sharing A is different from those not sharing A
- Clearly every system sharing *some* files is not willing to share *every* file

18

Directory Service Interface
Uniformity

- *Complete* uniformity is thus an attractive (maybe) theoretical idea that is nonsense in practice
- Uniformity of *subsets* of the name space is useful
  - » Shared software
  - » Shared data
- Large portions of many (most) systems will be common and should have the same structure for ease of management
  - » /usr/local on Unix machines
- Common structure will commonly allow for a private area
  - » /users/foo or /projects/bar
- Non-uniform, non-universal portions are still common shared among a subset of users

19

Directory Service Interface
Uniformity

- Standard challenges related to maintaining a consistent global view of a shared/distributed data set
  - » How to support the semantics of operations(which generally give an *unshared* view of the data and operations) on elements of the shared data set
- System must support a notion of elements that are composed to form the name space for any given machine
  - » File systems (partitions) are generally the elements
  - » *mount* is the operator for composition
- Hosts supporting a partition *export* it, making it available for sharing
  - » Sometimes globally, sometimes to a specific set of machines

20

Directory Service Interface
Name Space Structure

- Both users and administrators must be well served
  - » Sometimes conflicting goals: simplicity vs. transparence
- Three common approaches
  - » Machine + Path: /machine/path
  - » Mounting remote file systems onto the local file system
  - » Transparent symbolic links to non-transparent names
- /usr/local on many networked UNIX machines is a symbolic link to /net/server/d4/rtools/...
  - » This provides a (thin) layer of information hiding
  - » Single name space appears the same on all machines

21

Directory Service Interface
Name Space Structure

- Requirements and methods are still evolving
  - » What is desirable and *especially* what is cost effective are still open issues
- Common software and public information commonly shared
- Security and privacy of other modes of use is not as clear
  - » Trusting work groups is the most common mode

22

Semantics of File Sharing

- When two or more processes share the same file
- Semantics of reading and writing by each party must be defined precisely
  - » Primarily this relates to when changes made by each party are
    - ❖ Reflected in the file
    - ❖ When they become visible to the other party
- Several possible approaches
  - » UNIX semantics
  - » Session Semantics
  - » Immutable Files
  - » Atomic Transactions

23

Unix Semantics

- Every operation on a file is instantly visible to all parties
- A Read following a Write will return the value just written
  - » For all users of the file
- Enforces (requires) a total global order on all file operations to return most recent value
  - » On a single physical machine this results from using a shared I-Node to control all file operations
  - » File data is thus shared data structure among all users
  - » Distributed file server must reproduce this behavior
    - ❖ Performance implications of “instant updates”
    - ❖ Fine grain operations increase overhead

24

Unix Semantics

- Distributed UNIX Semantics
  - » Could use a single centralized server which would thus serialize all file operations
    - ❖ Provides poor performance under many use patterns
- Performance constraints require that the clients cache file blocks, but the system must manage consistency among cached blocks to produce UNIX semantics
  - » Writes invalidate cached blocks
  - » Read operations on local copies "after" the write according to a global clock happened "before" the write
    - ❖ Serializable operations in transaction systems
    - ❖ Global virtual clock orders on all writes, not reads

25

Session Semantics

- UNIX semantics are still expensive
  - » Write invalidation of all cached blocks slows write operations and reduces read performance
  - » Relaxation of the file interaction semantics helps
  - » Make changes to local copies and propagate them when the file is closed
- Session semantics because the changes become visible when the session is finished
- Final file state depends on who closes last
  - » OK for processes whose file modification is transaction oriented, open-modify-close
  - » Very Bad for mode of open for a series of operations

26

Session Semantics

- Semantics could arbitrarily chose update order
  - » No real guidelines or obvious reason to formulate a rule
  - » Modification of file by a process is monolithic
- Violates the familiar UNIX semantics implied by a single file pointer shared among parents and children
  - » Two processes appending to a file should produce cumulative results interleaved by write operation order
  - » Session semantics would produce one process's changes or the other, not both
    - Many processes keep files open for long periods
- Usable with caution but differs from many programmers' previous experience, so must be approach with caution

27

Immutable Files

- No updates are possible
  - » Simplifies sharing and replication
- No way to open a file for writing or appending
- Only directory entries may be modified
- Create a new file to replace an old one
- Also fine for many applications
  - » Again, though, different enough that it must be approached with caution
- Design Principle:
  - » Many applications of distribution involve porting existing non-distributed code along with its assumptions

28

Atomic Transactions

- Changes are all or nothing
  - » Begin-Transaction
  - » End-Transaction
- System responsible for enforcing serialization
  - » Ensuring that concurrent transactions produce results consistent with some serial execution
  - » Transaction systems commonly track the read/write component operations
- Familiar aid of atomicity provided by transaction model to implementers of distributed systems
  - » Commit ad rollback both very useful in simplifying implementation

29

Distributed File System Implementation

- General Design Principle:
  - » Design the system to handle how it is *actually used* well
  - » RISC argument after years and years of CISC
- File Use
  - » Results of studying actual file use can be surprising
  - » File use also changes with changing applications
    - ❖ Audio, Video, Graphics increase frequency of large files
- Most files are under 10 KB
  - » Could mean full file transfer fine for most situations
  - » May be changing with changing applications
    - ❖ Perhaps a multi-modal distribution

30

Distributed File System Implementation

- Most files have a short lifetime
  - » Create, read, delete
    - Temporary files for compilers and other programs
    - Could easily be local to a client
- Few files are shared (concurrent access)
  - » Client caching is fine for single user
  - » Session semantics are fine
    - Impose an overhead in unusual (concurrent access) case
    - Possible conversion of semantics when concurrent access is initiated

31

Distributed File System Implementation

- There are distinct classes of files
  - » Different lifetimes, uses, and preferable semantics
- Executable Files
  - » Needed everywhere, but rarely change
  - » Wide replication is fine
    - Complicates occasional update, but so what?
- Compiler and other temporary files
  - » Short lifetimes and often short files
  - » Unshared
  - » Easily and optimally kept local to the client

32

Distributed File System Implementation

- Mailboxes are frequently updated but rarely shared so replication is unlikely to help performance
- Ordinary data files may well be shared and accessed concurrently
  - » Many readers, single writer is common and best handled differently from many concurrent writers
- Conclusion: There are many classes of files and many types of access requiring a variety of types of support
- System software can support many of these and even choose among them based on usage pattern and locations of users and files

33

System Structure

- Several choices about how file servers and directory servers can be structured
- Are Clients and Servers different?
  - » Many systems make no distinction and can be both since they all run the same software
    - ❖ NFS remote file client and server
  - » Client and Server could be just user programs
    - ❖ Also making support of both easy
  - » OS software and even hardware may be different
    - ❖ File Server machine configurations and even embedded configurations (Network Appliances Boxes)

34

System Structure

- Directory Service and File Access Service
  - » Single server combines essentially separate functions resulting in more complex software
  - » Separate servers results in more communication
- Separate Servers
  - » Path to Binary (machine:I-Node) translation by directory service
  - » Binary name then used to gain access through file server
  - » More flexible and simpler software with more obvious structure
- Distribution can still give rise to complications

35

System Structure
Directory Services

- Consider a file system composed of multiple physically distributed elements
  - » One directory server per physical component
- Path name to binary file reference can be complicated
  - » Consider translation of /a/b/c
  - » Fig 5-7, page 260 Tanenbaum
- Each server responds to client with translation of the component that resides on its system
  - » More communication but ordinary RPC adequate
- Servers could forward requests that cross physical boundaries
  - » Less communication but smarter server software

36

System Structure
Caching

- Cache path translations on client to speed operation
  - » Files frequently used
  - » Frequent prefixes (/usr/local/bin)
- Cache misses default to basic lookup behavior
- Cache hits give binary file references
  - » BUT the reference may be *stale* so the file server must be able to reject such a reference and tell the client that it should do a regular lookup
  - » This is MORE expensive (latency and messages) so hints must be right most of the time

37

System Structure
State

- Should servers maintain state information about clients
  - » Advantages and disadvantages to both (of course)
- Stateless server advantages
  - » Inherently fault tolerant
    - Client crash doesn't really matter
  - » No OPEN/Close Messages
  - » No server data structures per call
  - » No open file limits
- Client requests are self-contained, increasing message length
  - » Every message must contain context
- File locking requires a special lock server

38

System Structure
State

- State aware server
  - » Must context information about every open file
  - » Data structure size and computation load
- Read/Write message are smaller
  - » Context ID rather than full file ID
- Client crashes leave an irrelevant context
  - » Classic problem: Distinguishing crashed and slow client
  - » Timeout too long: wastes space
  - » Timeout too short: invalidates inactive sessions
- Session and sequence numbers help track situation
- Read ahead and file locking also advantages

39

Caching

- Caching stores frequently or recently used data to improve performance, as usual
- Four potential places to store parts of a file
  - » Server Disk
  - » Server's Main Memory
  - » Client Main Memory
  - » Client Disk (if available)
- As usual, again, which is best depends on what is happening
  - » Best in different ways as well: performance, ease of implementation, simplicity

40

Caching

- Server's Disk
  - » Most straightforward
  - » Usually plentiful resource
  - » Accessible to all
  - » Poor performance and choice as information has farthest and longest to travel
- Server's Main Memory
  - » Keep recently use files in faster medium
  - » If server cache hits, server disk access avoided
    - Network transfer still happens and likely dominant
  - » Less plentiful than disk

41

Caching

- What Unit should we manage?
  - » Whole Files
  - » Disk blocks
    - Uses cache and disk space more efficiently
    - Better suited to unit of access and use of the resource
- Cache content replacement
  - » Victim selection algorithm
  - » Any basic caching algorithm is probably fine
    - Access is at a fairly coarse time scale
  - » LRU with linked lists probably feasible
- Evicted block written to disk if necessary or just evaporates

42

Caching

- Server side Main Memory is totally transparent to client
  - » But still requires network transfer
  - » Network latency typically the same order of magnitude as the disk
    - ❖ Bigger or smaller depending on disks and network
- Moving to main memory reduced disk influence
  - » Client side caching eliminates network influence
- Client side Disk caching
  - » Eliminates Network, but reintroduces disk
  - » Often plentiful but introduces coherency problems
  - » Usually more plentiful than main memory

43

Caching

- Client side main memory
  - » Eliminates both sources of large latency
  - » Still has cache coherency problems since there are now multiple copies of the disk block
  - » Most commonly used
    - ❖ Best cost/benefit ratio
- Several options about where to cache the information
  - » Figure 5-10, page 264 Tanenbaum
  - » Client Process memory
  - » Kernel Memory
  - » Cache manager process memory

44

Caching

- Client process memory
  - » Cache managed by system call library
  - » File written back to server when client exits
  - » Extremely low overhead
  - » Only effective if clients frequently open and close files
- Kernel Memory
  - » Disadvantage: always requires a system call even on hits
  - » Cache surviving across process exit more than compensates
  - » Consider a two pass compiler one pass writes and the other reads an intermediate file

45

Caching

- Separate User-Level Cache manger process
  - » Keeps kernel free of distributed file system code
  - » Easier to program and run experiments
    - Isolated
    - Well defined
  - » Cache pages could be paged out (user level VM)
    - Defeats the purpose
    - Could lock cache pages into main memory

46

Cache Consistency

- As usual you cannot get something for nothing
- If two clients access a file concurrently and both read, there is not problem, but if both *write* we could handle it by
  - » A third client seeing the original file, not the modified version
    - ❖ We could eliminate this problem by adopting session semantics
  - » This simply redefines incorrect behavior as correct
  - » If you or your program expects UNIX semantics this is wrong
  - » Or, the last client to write the file back could have its changes preserved - basically session semantics

47

Cache Consistency
Write Through Solution

- Does not affect (reduce) write traffic
  - » Simple and effective
- When a page is modified the new value is kept in the cache
  - » Also *written through* to the server
  - » Wasteful for successive writes with no intervening read
- A new process accessing the file sees the new values
  - » Still has problems
- Suppose A write a file and terminates
  - » A machine still has cached file
  - » B reads and modifies the file writing it back
  - » New process on A reading the file gets the old contents

48

Cache Consistency
Write Through Solution

- Could have a client check with the server before using cache when a new process opens the file
  - » Epoch number associated with the file version
  - » Form of global event (version) ordering
- Requires an RPC, but transfers a small amount of data
  - » Validates a large amount of data (cache) for use
- Write-Through reduces on read traffic
  - » Write traffic is the same (write-through)
- Could cheat by only noting that the file has been modified and then either demands changes when sharing begins or periodically collects updates
- What does a second process see on open - depends on when

49

Cache Consistency
Write on Close

- A next step is to match session semantics
  - » Write the file only after it has been closed
- Even better, wait for a timeout period after it has been closed to see if it will be reopened or deleted before sending it to serer
- Still allows a second process to overwrite the first process's modifications, but that is true of all session semantics local or distributed

50

Cache Consistency
Centralized Control Algorithm

- UNIX semantics but not robust and scales poorly
  - » When a file is opened a message is sent to server
  - » Server tracks files open for reading and/or writing
  - » Multiple readers are permitted
  - » Single writer constraint enforced
  - » File close write the file back to the server
- Requests for an open file may be granted or denied
- Alternatively the server may send a message to clients
  - » Invalidating/flushing the cached copies and disabling caching
    - This allows multiple readers/writers
  - » Dynamic semantics

51

Cache Consistency
Centralized Control Algorithm

- Inelegant since it involves unsolicited message from the server to the client
- Server still must check to see if a cached file is valid and the requested access is permitted

52

Replication

- Additional service that a distributed file system can provide
  - » Multiple copies on different disks and servers
- Advantages
  - » Increases *reliability* since failure of one copy does not destroy the file
  - » Increases *availability* since one server being busy or down does not deny access tot he file
  - » Increases *performance* since the system can spread file server workload across multiple servers
- Disadvantages
  - » Increased overhead for storage and administration

53

Replication

- Key Issue: Transparency
  - » Full range of treatment
  - » Full user knowledge and management
  - » Complete system concealment and transparency to user
- Three major approaches
  - » User management
  - » Lazy replication
  - » Group communication support
  - » Figure 5-12, page 269 Tanenbaum

54

Replication
User Management

- Client program of distributed file service explicitly creates and manages multiple copies
  - » Distributed file system sees each as an independent files
- Directory server might still support the notion of multiple copies and be able to return multiple references
  - » Assumes that client code and directory service use the same conventions and/or that the client informs the directory server through its interface
- Attractive for implementing application specific fault tolerance and management semantics
  - » Source code control
  - » Data bases

55

Replication
Lazy Replication

- Client creates a single copy by interacting with some server
  - » Server conceals replication semantics
  - » Separates replication and application semantics
- Server holding the original copy is the interface and must be smart enough to retrieve other copies at need
  - » Limited use when original server fails
  - » Advantage of being transparent to user
  - » Server fault tolerance could be introduced through name transparency
- Server layer also assume responsibility for coherence of the multiple copies
- Delay in creating the copies is also a vulnerability

56

Replication
Group Communication

- Combines aspects of User and Lazy approaches
  - » Provides replication support at the client side since this is where the group communication is initiated
  - » Transparency to user still substantial since the multiplicity is concealed inside the WRITE calls
- Client retains ability to describe replication and thus fault tolerance semantics by describing group communication
- Advantage:
  - » Group approach concurrent with user actions
  - » Lazy replication creates copies in the background, often after user finishes creating a new copy of file
    - Session semantics flavor

57

Replication
Design Issues

- Which approach is better
  - » It Depends
    - OK, on what?
- Application level semantics have a large influence
  - » Replication supports issues whose importance and cost/benefit tradeoff vary widely with application
  - » Fault tolerance
  - » Reliability
  - » Availability
- Transparency is difficult because of this wide variance in application semantics and cost

58

Replication
Update Protocols

- How can existing replicated files be modified?
  - » Coherence requires atomic update semantics for copies
- Sequential update messages to servers holding each copy cannot support coherence through atomic update
- Two major approaches
  - » Primary copy
  - » Voting
- Primary copy approach designates copy on one server as primary making all others secondary
  - » Client updates only the primary
  - » Primary sends all updates to the secondary copies
  - » Reads can be done from any server

59

Replication
Update Protocols

- Primary server fault tolerance
  - » Write update messages to stable storage (log) before updating the file
  - » Makes updates atomic and recoverable across server crashes since any in progress can be restarted on reboot
- Single point of failure
  - » If the primary server is down → no updates
  - » Election algorithms might help
    - Promote a secondary to primary
    - Directory service must hold enough group information
- Voting approach addresses primary failure

60

Update Protocols
Voting

- Client must get permission from multiple servers before reading or writing a file
  - » More robust since any server being down not fatal
- Update example:
  - » Client contacts a majority of servers to approve update
    - Analogy to two-phase locking and transactions
  - » Agreement of a majority commits the update creating a new version number of the file
    - Dissenting servers will eventually discover the majority decision and concur
- Reading: client determines current version by receiving the same version number from a majority

61

Update Protocols
Voting

- Subtle(?) semantic point
  - » What if a reader determines the most recent version
  - » Then another client begins an update
  - » Can they be concurrent
    - Yes if servers preserve all active versions and serialized transaction semantics are acceptable
    - Write will produce results consistent with having happened "after" the read
- Gifford's approach defines the idea of a *quorum*
  - » Separate number of servers for read N_r and write N_w
  - » Decisions are consistent when $N_r + N_w > N$

62

Update Protocols
Voting

- Interesting tradeoffs between read and write latency addressed by various values of N_r and N_w
  - Roughly equal requires each to contact $N/2(+1)$ servers
  - $N_r=1$ minimizes read latency by requiring that a client only find and contact one sever with a copy of the file
    - Requires $N_w=N$ maximizing write latency
    - Appropriate for many readers, single writer situations
- Voting with ghosts handles a problem when N_r is small
  - N_w is large and when servers are down is may not be possible to form a write quorum
  - Ghosts represent down servers and can participate to form a write quorum

63

Update Protocols
Voting

- Voting with ghosts works because N_r and N_w are chosen to ensure that reading and writing are mutually exclusive
- Interesting tradeoffs between read and write latency addressed by various values of N_r and N_w
  - Roughly equal requires each to contact $N/2(+1)$ servers
  - $N_r=1$ minimizes read latency by requiring that a client only find and contact one sever with a copy of the file
    - Requires $N_w=N$ maximizing write latency
    - Appropriate for many readers, single writer situations
- Servers represented by ghosts must obtain the most recent version when they come up
  - Always true

64

Network File System

- Created by Sun Microsystems and probably the single biggest distribution success story in computing so far
  - » BUT carries with it baggage of history
  - » Created for significantly different computing context
- Supports heterogeneous systems
  - » CPU
  - » Data format
  - » Networks
- NFS architecture allows any machine to be both a client and a server
  - » Each machine exports directories offered for remote use

65

Network File System

- Entire directory sub-trees are exported
  - » NFS server offers the *mount point* for export
    - Simplifies accounting and implementation because the systems need track only a single directory for each exported file system
    - Constrains name space structure and sharing semantics but this seems easy enough to handle in practice
  - » Each system announces exports in /etc/exports
- Clients import directories by mounting them in the local name space
  - » Crossing the mount point during name to I-Node translation signals file system type and protocol change

66

Network File System

- Flexible and general mechanism with several good points
  - » Isolates distribution to a small part of the client system
    - New file system and/or I-Node type noting distribution
  - » Even diskless workstations can do this, mounting their root FS
  - » Creates substantial transparency of local/remote file location and thus enables workstations to have the same functional structure while occupying any point in a range of local/remote file
    - Local and remote file location affect only performance
    - Graceful transition between local disk and diskless

67

Network File System
Protocols

- Standard protocols are required to enable support for heterogeneous systems
  - » Experience has shown that *open* standards almost always win against closed standards
    - Eventually
    - Small sample size
- A protocol is a set of requests a client can make and the corresponding replies that a server should make
  - » Defines and limits interaction semantics
- NFS defines two client-server protocols
  - » Mounting
  - » Directory and file access

68

Network File System
Mount Protocol

- A client can send a request to mount a particular directory path name to a server
  - » If the path name is a legal directory and has been exported then the server returns a *file handle*
- The file handle contains several kinds of information
  - » File system type
  - » Disk
  - » I-Node number
  - » Security information
  - » Note: client already knows the server
- Remember: file handle is a *magic cookie*

69

Network File System
Mount Protocol

- Systems are often configured to automatically mount sets of remote directories at boot time
  - » This can cause problems when remote servers are down
    - Delay or hanging
  - » Wastes time, system, and network resources when the directories are not actually required
    - NFS mount tables are the *superset* of all information than *can* be used
- Auto-mount created to address this problem
  - » Remote directories are mounted only when requested in the course of a name to I-node translation
  - » Allows a set of remote directories to be associated with the mount point

70

Network File System
Mount Protocol

- First of associated servers to reply is used
- Simple race between servers is also a simple load balancing mechanism
  - » First to reply has the lowest latency (modulo skew in request transmission times)
  - » System with the lowest latency is likely to be the least loaded
    - Assuming network delay does not dominate
    - Lowest network latency is also desirable, but different
- NFS does not explicitly support replication so USER is responsible to ensure that multiple mount points are identical → good for read-only file systems

71

Network File System
Access Protocols

- Clients send massages to servers to access and manipulate
  - » Files
  - » Directories
- Most UNIX file and directory library and system calls are supported
  - » Substantial transparency and distributed support for the basic file system structure and semantics
  - » EXCEPT OPEN and CLOSE
- LOOKUP message finds information about a file, including its handle, but does not maintain internal tables
  - » Why? What are the implications

72

Network File System
Access Protocols

- NFS is a *stateless* protocol
  - » Stateless servers are simpler and more robust
  - » Require that each message be self-contained
    - ❖ Contain all information required to satisfy request
- Read requests contain
  - » File handle
  - » Offset
  - » Number of bytes
- Stateless server can crash and reboot (quickly) without client even noticing
  - » No information is lost since none is maintained

73

Network File System
Access Protocols

- Design Principle:
  - » Design servers to be stateless if possible
  - » Design servers to have a stateless component if possible
- Stateless benefits are numerous but is in conflict with some aspects of UNIX file semantics
  - » File locking is an aspect of *file state*
  - » Stateless NFS context requires and additional mechanism to preserve the state
    - ❖ Lock server
    - ❖ Optimize common case

74

Network File System
Access Protocols

- Security
  - » Originally NFS used basic *rwx* bits for owner group and world
  - » Access messages contained the user and group numbers of the client
    - ❖ Naïve and easily spoofed
    - ❖ Write your own NFS client filling in and ID numbers you like
    - ❖ NFS often specifically excludes super user access
  - » Public Key cryptography can be used (optionally) to validate client and server on each request/reply
    - ❖ Data not encrypted

75

Network File System
Implementation

- Sun's NFS implementation has 3 layers
  - » System call
  - » Virtual File System
  - » Local or NFS file system component
  - » Figure 5-14, page 276 Tanenbaum
- System calls
  - » Handles calls associated with file model: open/close. Read/write, seek/tell
  - » Parses call parameters and checks for errors
  - » Translates and prepares the invocation of a corresponding part of the VFS

76

Network File System
Implementation

- Virtual File System
  - » Abstracts the basic file model while wrapping calls to a specific file system
    - ❖ Precursor to file system switches and myriad of file system types in Unices and NT
- VFS maintains a table with one entry for each open file
  - » Analogous to the system file table and set of open I-Nodes associated with the local file system
  - » The V-Node (virtual I-Node) represents each open file and indicates if it is local or remote
    - ❖ Holds enough information to access the item

77

Network File System
Implementation

- Example: mount, open, read/write, close sequence
- Mount:
  - » Specifies local directory mount point and remote directory to mount upon it
  - » Asks remote server for a *handle* to the directory
    - ❖ Returned if exists and exported
  - » Makes a *mount* system call passing handle to OS
  - » Kernel constructs a V-Node to refer to the remote directory and asks the NFS client to create a R-Node to hold the handle
    - ❖ V-Nodes refer to I-Nodes (local) or R-Nodes (remote)

78

Network File System
Implementation

- Open:
  - » Path to I-Node translation is now path to V-Node
  - » Finds that the name crosses a remote mount point and will thus find the reference to the mounted R-Node
  - » Asks NFS client to send path name suffix (after remote mount point) to the server for translation to file handle
  - » Client stores the file handle in the R-Node and returns R-Node reference to VFS layer
  - » User is given a file descriptor which is mapped to the V-Node created
- Local data structures contain information required to do file operations

79

Network File System
Implementation

- Read:
  - » Client-server operations use 8K messages
    - Even when less is required
  - » VFS layer automatically issues read-ahead request
    - Requests next 8K chunk when it receives a request for one
    - Optimization of common sequential access case
    - Promotes *concurrent* server execution
- Write:
  - » Locally buffered until 8K chunk is accumulated
  - » Interaction with user level *file pointer* buffering
- Close: sends all client data to server immediately

80

Network File System
Implementation

- Interesting Issues and Factoids
  - » Servers maintain a main memory disk block cache to lower access latency
  - » Clients maintain two caches
    - ❖ File attributes (I-Nodes)
    - ❖ File data
  - » Caches require coherency control
    - ❖ NFS uses 3 second data and 30 second directory timers
    - ❖ Open operation on a local cached file issues a concurrent server update time query
    - ❖ 30 second timer flushes all modified blocks to server

81

Network File System
Implementation

- Caching causes the UNIX file semantics to be distorted
  - » File modifications may only be visible after 30 seconds
  - » Concurrent writes to a single remote file from different machines do not have a well defined result
    - ❖ Race condition
- Empirical Observation:
  - » Complete transparency is often expensive or impossible
  - » Translucence *often* useful for most applications
  - » You *must* know the difference
- NFS popular BUT not appropriate for naïve distribution of multi-process applications depending on concurrent file access semantics

82

Distributed File Systems Lessons Learned

- Observations in 1990 by Satyanarayanan about distributed system design
- Workstations have cycles to burn
  - » Best price/performance ratio
  - » Distributed and scalable resource (not server)
  - » Owned by person requesting remote service
- Cache whenever possible
  - » Even modest caches can save large amounts of system, network and shared resources
  - » Be cautious about coherency implications
    - Evaluate costs as well as benefits

Distributed File Systems Lessons Learned

- Exploit usage patterns
  - » Optimize important special cases
    - Frequent
    - Easy
  - » Balance against complexity of too many methods for the same basic operation
- Minimize system-wide knowledge and change
  - » Constrains concurrency
  - » Increases latency
  - » Increases management complexity
  - » Affects scaling
  - » Attraction of stateless servers and hierarchic designs

Distributed File Systems Lessons Learned

- Trust the fewest possible entities
  - » Lowers risk
  - » Lowers verification overhead
  - » Affects scaling
    - ❖ Avoid per-workstation dependency
- Batch work when possible
  - » 8K file block operations is an example
  - » Increases latency and coherency concerns
- Common Design Challenge
  - » Desirable goals are in competition and thus final design must compromise among them

85

Trends in Distributed File Systems

- Hardware
  - » Costs continue to drop at an amazing rate
- Rapidly dropping memory costs make it possible to have every larger data bases in main memory
  - » Servers could have entire data set in main memory
- Still a cache, so coherence problems with stable storage still exist
  - » Write-through policy
  - » Idle cycles used to write to disk
- RAM disk and file system
  - » IDE Flash-ROM 20 MB disks

86

Trends
Hardware

- Write-Once Read Many (WORM)
  - » CD-ROM burners
  - » Excellent backup and archiving method
  - » Jukeboxes add a level to memory hierarchy
    - CD - Disk - Main Memory
  - » Increasingly cheap
  - » Still fairly slow
- Huge capacity networks
  - » 100 Mb/s and Gb/s radically change distribution and caching tradeoffs
  - » U of Washington use of idle remote workstation memory instead of disk as VM cache

87

Trends
Hardware

- Specialized hardware for sophisticated systems
  - » Real-time support
  - » Distributed synchronization and control
- FPGA synergy
  - » Consider modest FPGA assets in a workstation which could be made to do many things
  - » Distributed locking and cache block invalidation
  - » Special administrative ATM virtual circuits with group communication support
  - » Direct handling of administrative cells
    - Concurrent with CPU

88

Trends
Scaling

- Distributed system size strongly affects algorithm choice
  - » Working well for 100 machines means nothing for 10K
- Centralized algorithms do not scale well
  - » Often distributed ones do not either
    - ❖ Distributed mutual exclusion
  - » Partitioning and hierarchic organization often helps
- Broadcasts are a problem
  - » Consider CPU broadcasting one message per second
  - » N of these generate N interrupts at N machines
    - ❖ Not a problem for N=10
    - ❖ VERY problematic for N=10K

89

Trends
Scaling

- Data structures become important with scaling
  - » Linear search easiest and fastest for 10
  - » Self abuse for even 100
- Strict semantics are harder to implement as systems scale
  - » Design Principle: use weakest semantics that make sense
  - » Trade off ease of programming with scalability
- Name space
  - » How long can/should path names get?

90

Trends
Wide Area Networking

- Virtually all distributed system research has been done in the context of LANs
  - » Considerable changes with WAN context
  - » Latency
  - » Loss
  - » Cost
  - » Interaction
- WAN access of major economic importance
  - » WWW commerce
  - » Video on demand
  - » Distributed Virtual Environments

91

Trends
Wide Area Networking

- Commercialization will create many changes
  - » Service providers have no coherent pricing model
- Economies of scale
  - » IP phones
- Ubiquitous Mobile Environments
  - » Consistent interface
  - » Anywhere
    - Home, Office, Airport Kiosks
  - » Any Platform
    - Workstations to Palm Pilot

92

Trends
Mobility

- Network addressing is a big challenge - Mobile IP
  - » May be transparent to distributed computing level
- Often seen as highly variable communication bandwidth
  - » Isolated
  - » Wireless
  - » Wired
- Interesting effects on caching
  - » CODA file system claims to support mobility and intermittent connection
  - » Coherency on steroids
- Constraints on application semantics

93

Trends
Mobility

- Rapidly Deployable Radio Network - RDRN
  - » Wireless end-user and network nodes
  - » Steerable communication beams
  - » Self-organizing network structure
- Management software is clearly distributed
- Interesting distributed system issues
  - » Election: DNS server
  - » Local connection choices have global consequences
    - Iterative network topology adjustment?
  - » Shorter time scale on link state changes?

94

Trends
Fault Tolerance

- Most systems are not fault tolerant
  - » But the general population expects things to work
  - » Phone system → IP phones?
- Requires considerable redundancy
  - » Hardware
  - » Communication infrastructure
  - » Software
  - » Data
- File replication will become essential
- Systems must be designed to function with partial data
  - » Mobility

95

Trends
Fault Tolerance

- Down-times and periodic crashes will become less and less acceptable as computers spread to non-specialists and into commodity functions
  - » ATM machines
  - » Microwaves
  - » Phone system (IP mode)
- Expectations/abilities/costs are not well balanced
  - » People want more than is there but want it to cost less
  - » Potential brake on Internet and automation expansion

96

Trends
Multi-Media

- Current data files are rarely more than a few MB
  - » MM files can exceed GB
  - » Compression clearly popular because of this and has a fundamental affect on network requirements and economics
- Video-on-demand
  - » Significant affect on network traffic
  - » Perhaps also on file systems
  - » Real-time support is interesting as well

97

Trends
Virtual Environments

- Many observer's current "killer application" candidate
  - » Still no pricing model, so how do you make money?
- Extremely challenging distribution issues
  - » Communication patterns are dynamic
  - » Small messages
  - » Fine temporal scale
  - » Significant scalability potential
- Multiple senses
  - » Sound, touch, smell as well as vision

98

Summary

- Distributed file systems are central to many of the most popular uses of distributed systems
- Transparency is desirable but often hard to achieve
  - » How do distributed semantics differ from local
    - ❖ Subtly
    - ❖ Intermittently (race conditions)
- Name spaces must be adapted in some fashion
  - » Location transparency and independence
- Semantics should be weakened for performance and correctness vs. accidentally or implicitly
  - » File access, modification, locking
  - » Distributed model may be best original choice

99

Summary

- Session semantics and immutable files are attractive because of performance advantages
  - » Useful in many situations but importantly different
  - » Porting poses particular challenges to semantics
    - ❖ Sometimes depends on unpublished behavior
- Transactions are attractive
  - » Well structured
  - » Reversible
  - » Often overkill - high overhead
- Implementation means hard choices
  - » Simplicity vs. efficiency vs. scalability

100

Summary

- Stateless servers attractive
  - » Simple *and* efficient for many applications
    - WWW
- Caching has a huge affect on
  - » Performance
  - » Complexity
  - » Overhead
- Replication and fault tolerance will become increasingly important and ubiquitous
- NFS an instructive example
  - » Simpler than you might think but widely used