

CSSE4004-Lecture 9

Distributed File Systems

WWW

Distributed Coordination

Learning Objectives Distributed File Systems

- Naming
 - transparency – evaluate alternatives
- Synchronisation
 - models and trade-offs
- Replication, caching
 - Models and trade-offs
- Fault-tolerance
 - trade-offs again

2

Distributed File Systems- Key Issues

- Naming
- Performance
- Fault-tolerance
- Also important (not covered in this course)
 - Security
- File systems discussed:
 - NFS
 - Coda

3

NFS

Sun Network File System

- Designed for heterogenous use but UNIX-like
- Transparent access to remote files
 - remote access model – no local copy
- Virtual File System (VFS)
 - replaces UFS, can interface to other distributed FS

4

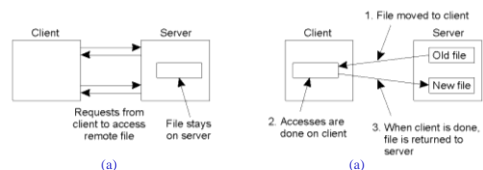
NFS Architecture...

RPC to communicate to server

- VFS hides nature of remote service
- UNIX-like **file handle**
 - obtain by opening file
 - use for subsequent file accesses
 - specify offset, number of bytes to read/write
- Not strictly stateless since version 4
 - server can remember file opened

5

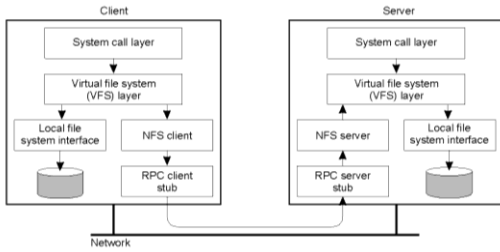
...NFS Architecture...



- a) Remote access model (NFS)
- b) Upload/download model (e.g., local edit of web page – or NFS with caches)

6

...NFS Architecture



Basic NFS architecture for UNIX systems

7

File System Model

| Operation | v3 | v4 | Description |
|-----------|-----|-----|---|
| Create | Yes | No | Create a regular file |
| Create | No | Yes | Create a nonregular file |
| Link | Yes | Yes | Create a hard link to a file |
| Symlink | Yes | No | Create a symbolic link to a file |
| Mkdir | Yes | No | Create a subdirectory in a given directory |
| Mknod | Yes | No | Create a special file |
| Rename | Yes | Yes | Change the name of a file |
| Rmdir | Yes | No | Remove an empty subdirectory from a directory |
| Open | No | Yes | Open a file |
| Close | No | Yes | Close a file |
| Lookup | Yes | Yes | Look up a file by means of a file name |
| Readdir | Yes | Yes | Read the entries in a directory |
| Readlink | Yes | Yes | Read the path name stored in a symbolic link |
| Getattr | Yes | Yes | Read the attribute values for a file |
| Setattr | Yes | Yes | Set one or more attribute values for a file |
| Read | Yes | Yes | Read the data contained in a file |
| Write | Yes | Yes | Write data to a file |

An incomplete list of file system operations supported by NFS (for the curious)

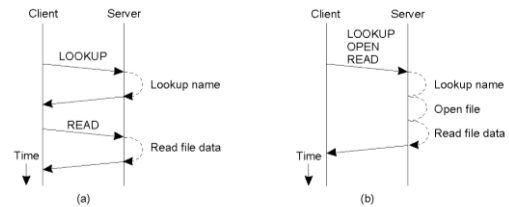
NFS Communication...

Open Network Computing RPC (ONC RPC)

- hides OS-dependent RPC features
- NFS version 4 supports *compound procedures*
 - group multiple RPCs
 - save latency of multiple RPCs
 - handled in order as if sent separately
 - if 1 fails, rest terminate

9

...Communication



- a) Reading data from file in NFS version 3
 b) Reading data using compound procedure in version 4

10

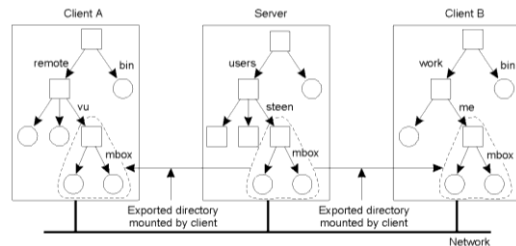
NFS Naming...

Transparent access to remote file system

- Mount *exported* (part of) file system
- Naming dependent on local directory
 - can't e.g. advertise path independent of machine
 - no *mechanism* for global consistency
 - *policy* common
 - standard local paths from root e.g., /usr/bin
- Works well in one LAN

11

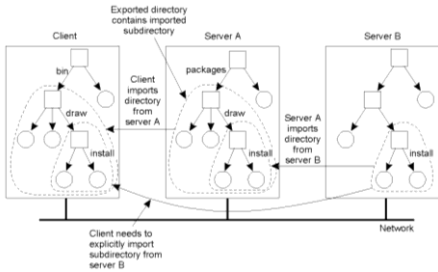
...Naming...



Mounting (part of) remote file system in NFS

12

...Naming



Mounting nested directories from multiple servers in NFS

13

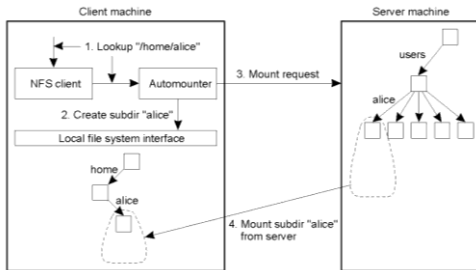
NFS Performance

Optimizations on top of basic architecture

- Since version 4 dropped statelessness
 - can implement client caching
 - server can initiate client callbacks
 - more practical on wide-area network
- Automounting
 - logically mount whole directory tree
 - only mount subtrees on demand

14

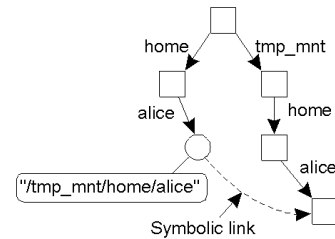
Automounting...



Simple automounter for NFS

15

...Automounting

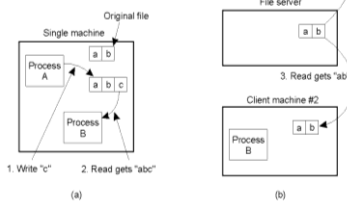


symbolic link to mounted directory prevents another automounter invocation once mounted

16

Semantics of File Sharing...

- On single machine, when *read* follows *write*, value returned by *read* is value just written
- In distributed system with caching, obsolete values may be returned



(a)

(b)

...Semantics of File Sharing

| Method | Comment |
|-------------------|--|
| UNIX semantics | Every operation on a file is instantly visible to all processes |
| Session semantics | No changes are visible to other processes until the file is closed |
| Immutable files | No updates are possible; simplifies sharing and replication |
| Transaction | All changes occur atomically |

4 ways of dealing with shared files in distributed system:
NFS uses *session semantics*

18

File Locking in NFS...

| Operation | Description |
|-----------|--|
| Lock | Creates a lock for a range of bytes |
| Lockt | Test whether a conflicting lock has been granted |
| Locku | Remove a lock from a range of bytes |
| Renew | Renew the lease on a specified lock |

NFS version 4 operations added file locking

19

...File Locking in NFS share reservations – implicit locks

| | | Current file denial state | | | |
|----------------|-------|---------------------------|---------|---------|------|
| | | NONE | READ | WRITE | BOTH |
| Request access | READ | Succeed | Fail | Succeed | Fail |
| | WRITE | Succeed | Succeed | Fail | Fail |
| | BOTH | Succeed | Fail | Fail | Fail |

(a)

| | | Requested file denial state | | | |
|----------------------|-------|-----------------------------|---------|---------|------|
| | | NONE | READ | WRITE | BOTH |
| Current access state | READ | Succeed | Fail | Succeed | Fail |
| | WRITE | Succeed | Succeed | Fail | Fail |
| | BOTH | Succeed | Fail | Fail | Fail |

(b)

The result of *open* operation with share reservations in NFS

a) When client requests shared access given current denial state

b) When client requests denial state given current file access state

20

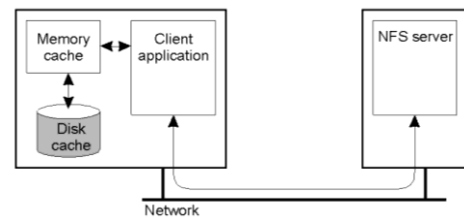
Client Caching...

Another violation of statelessness

- Added to version 4
 - version 3 had an outside protocol
 - version 4 included it
 - consistency implementation-dependent
 - more practical on wide-area network
- Server may *delegate* some rights to open file
 - local attempts at file locking handled locally
 - server can *recall* delegation

21

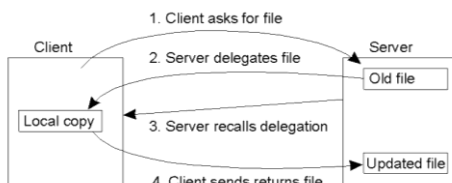
...Client Caching...



Client-side caching in NFS

22

...Client Caching



Using the NFS version 4 callback mechanism to recall file delegation

23

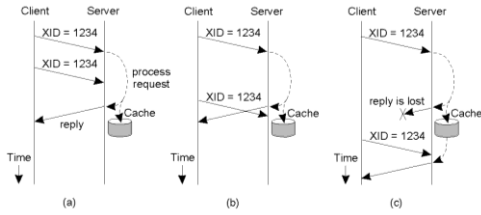
Fault Tolerance

Cost of adding statefulness in version 4...

- Specific state
 - file locking
 - delegation
- Unreliable RPC
 - duplicate requests
 - sequence numbering

24

RPC Failures



- request still in progress
- reply just been returned
- reply some time ago, but was lost.

25

NFS Summary

- Transparency
- No distributed naming policy built in
- Performance undone original stateless design
- Fault tolerance bit of an afterthought

in wide use despite limitations

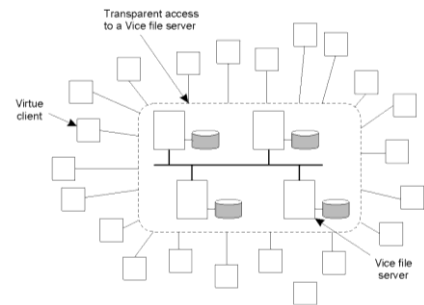
26

Coda File System

- Grew out of AFS version 2 (all from CMU)
 - designed to support entire CMU community
 - $\approx 10,000$ workstations
- **Vice** central file servers
- **Virtue** workstations
 - user-level process **Venus** (like NFS client)
- Looks like UFS through VFS
- Global shared namespace starting at /afs

27

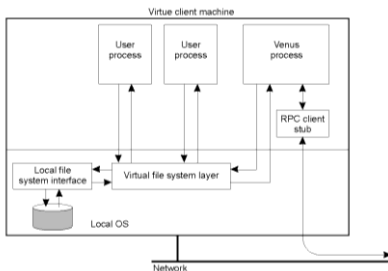
Overview of Coda...



Overall organization of AFS

28

...Overview of Coda



Internal organization of Virtue workstation

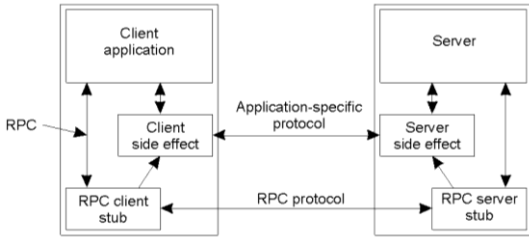
29

Communication...

- RPC2
 - Reliable RPC on top of UDP
 - *side-effect*: application-specific protocol
 - multicasting support
- MultiRPC
 - Send multiple RPCs in parallel

30

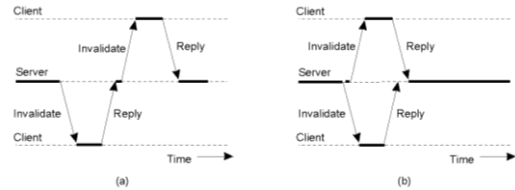
...Communication...



Side effects in Coda's RPC2 system

31

...Communication



- a) Sending invalidation message one at a time
- b) Sending invalidation messages in parallel

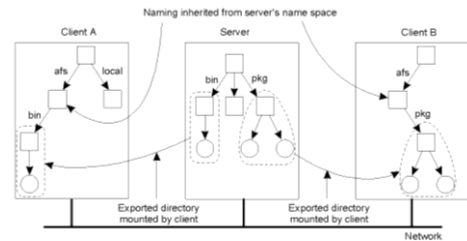
32

Naming...

- Some similarity to NFS but
 - shared name space maintained
 - all clients see same name if can see same file
 - support for finding instance of replicated file
- Overall effect very different

33

...Naming



Clients in Coda have access to single shared name space

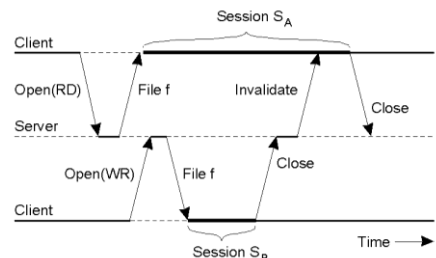
34

Synchronization

- Transactional semantics for high availability
 - locally cached files allow operation across failure
- Sharing
 - make local copy on open
 - others' write attempts fail if open already for write
 - different kinds of session

35

Sharing Files in Coda



Transactional behavior in sharing files in Coda

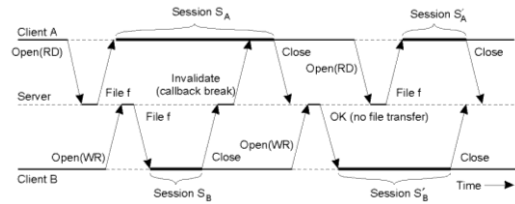
36

Caching and Replication

- Client-side caching
 - important for performance
 - callbacks to maintain consistency
 - *callback promise* – server knows client has file
 - *callback break* – after server asked to invalidate copies
- Server-side replication
 - *optimistic replication* – may be inconsistent
 - *versioning* used to make consistent

37

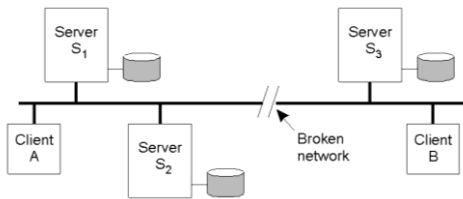
Client Caching



Use of local copies when opening session in Coda

38

Server Replication variant of Read-One, Write-All used



2 clients with different AVSG for same replicated file
(AVSG – Accessible Volume Storage Group)

39

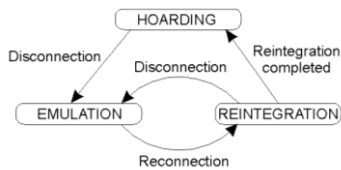
Fault Tolerance

Designed for high availability

- Disconnected operation
 - loss of contact with server: continue
 - on reconnection resolve conflicts
 - seldom problem needing manual intervention
 - write-sharing rare
 - *hoarding* – cache more files than immediately needed
- Recoverable virtual memory
 - files needed across crashes mapped into RAM

40

Disconnected Operation



State-transition diagram of Coda client with respect to volume

41

Summary

| Issue | NFS | Coda |
|-------------------|---------------------|-------------------------|
| Design goals | Access transparency | High availability |
| Access model | Remote | Up/Download |
| Communication | RPC | RPC |
| Client process | Thin/Fat | Fat |
| Server groups | No | Yes |
| Mount granularity | Directory | File system |
| Name space | Per client | Global |
| File ID scope | File server | Global |
| Sharing sem. | Session | Transactional |
| Cache consist. | write-back | write-back |
| Replication | Minimal | ROWA |
| Fault tolerance | Reliable comm. | Replication and caching |
| Recovery | Client-based | Reintegration |

Comparison between NFS and Coda

- Reading: Chapter 11 (without 11.1.2, 11.1.3, 11.3.3, 11.6.3-4, 11.7-8)

42

Learning objectives WWW and Distributed Coordination

- Synchronisation, replication and fault tolerance in WWW
- Rationale for coordination
- Linda mechanism in Jini
- Synchronisation, replication and fault tolerance in Jini

43

Contents

- Document-centric systems
 - WWW
 - Synchronisation and replication
 - Some fault tolerance achieved by caching and replication
- Coordination-based systems
 - Introduction to coordination systems
 - Linda model
 - Jini coordination

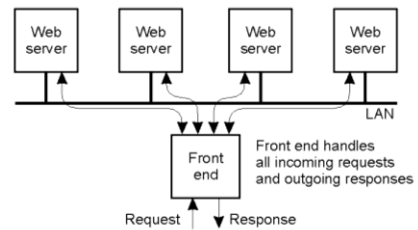
44

WWW

- Client – server architecture
 - Client/browser fetches documents
 - Some processing possible on
 - server side (e.g. CGI, server side JavaScript, servlet), or
 - client side (e.g. applets, client side JavaScript)
 - HTTP protocol for sending requests to server
 - HTML or XML for document definition
 - Naming: Uniform Resource Locators
 - DNS based
 - global naming

45

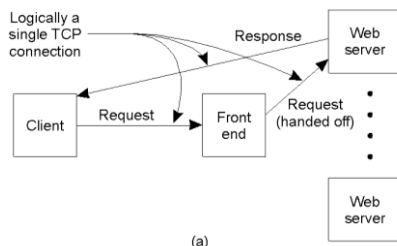
Replication - Server Clusters



The principle of using a cluster of workstations to implement a Web service (horizontal distribution)

46

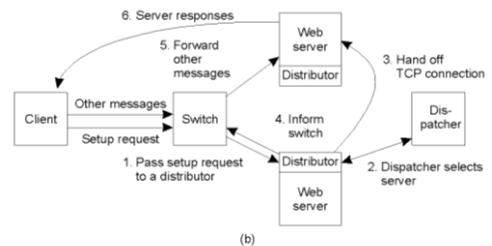
Server Clusters



(a) The principle of TCP handoff

47

Server Clusters



(b) A scalable content-aware cluster of Web servers

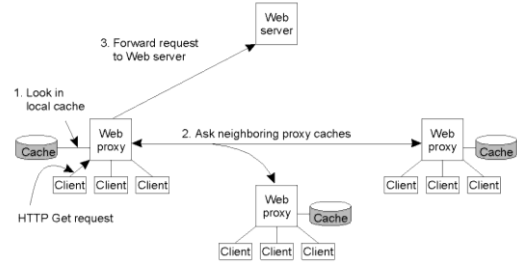
48

Synchronisation and replication

- Synchronisation does not exist in WWW
 - HTTP extensions proposed to provide synchronisation (e.g. for collaborative authoring)
- Replication
 - Client side caching
 - browser caching
 - proxy caching
 - Server replication
 - server clusters (transparent for users)
 - mirrors
 - content delivery networks

49

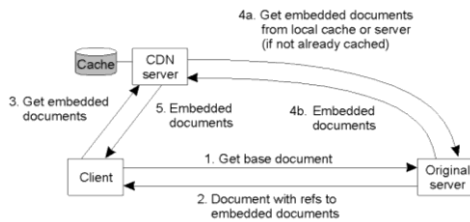
Web Proxy Caching



The principle of cooperative caching

50

Content delivery networks



The principle working of the Akami CDN.

51

WWW Summary

- Document based distributed system
- Minimal support for coordination
 - This may change in future
- Various types of replication
- Fault tolerance
 - Reliable communication
 - Server clusters to increase availability

Read 12.2.3, 12.5, 12.6, and have a look at other sections

52

Distributed Coordination

Components distributed: how to work together?

- Assume distributed process model
- Separate out
 - communication
 - cooperation

53

Introduction to Coordination

How closely are processes coupled?

- Temporally
 - know timing of communication
- Referentially
 - know who you're talking to
- High latencies, making coordination distinct:
 - temporally and referentially uncoupled processes
 - generative communication in Linda

54

Introduction to Coordination Models

| | | Temporal | |
|-------------|-----------|------------------|--------------------------|
| | | Coupled | Uncoupled |
| Referential | Coupled | Direct | Mailbox |
| | Uncoupled | Meeting oriented | Generative communication |

A taxonomy of coordination models (adapted from [cabri.2000])

55

Introduction to Coordination

- Direct coordination
 - explicit referencing, transient communication
- Mailbox coordination
 - Explicit referencing, persistent communication
- Meeting oriented coordination
 - Referentially uncoupled, transient communication (usually event based, publish/subscribe systems, e.g. Elvin)
- Generative coordination
 - Shared persistent dataspace of tuples

56

Examples

- Linda
 - introduced tuple spaces
- Jini
 - uses tuples for distributed coordination

57

Linda

Tuple spaces general mechanism for parallelism

- Tuple
 - ≥ 0 typed fields, e.g., (42, "Answer")
- Template
 - matched to tuple, e.g., (?int, "Answer")
- Operations
 - out – place tuple in tuple space
 - rd – copy matching tuple (block if none)
 - in – remove matching tuple (block if none)

58

Linda Challenges

Scalability and efficient matching

- Distribute tuple space
 - finding right match even more challenging
- Efficient matching
 - ideally fast associative store
 - in practice, some kind of fast indexing

59

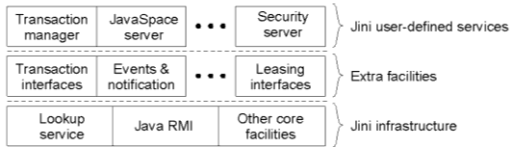
Jini

Focus here on JavaSpaces – based on Linda

- Shared space containing typed tuples
 - serialized – marshaled
- Matching using templates
 - match data values
 - all match NULL

60

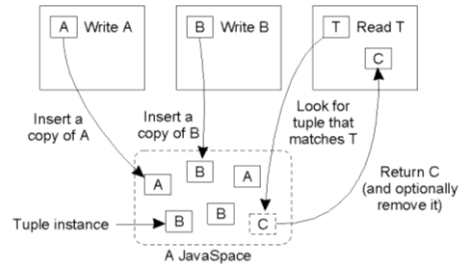
Architecture



The layered architecture of a Jini System.

61

Overview of JavaSpaces



General organization of JavaSpace in Jini

62

Jini operations on JavaSpaces

Minor renaming of Linda operations

- write
 - serialize and place in tuple space
- read
 - if match to template, return value of tuple
 - if no match block
- take
 - like read but remove from tuple space
- non-blocking versions of read and take

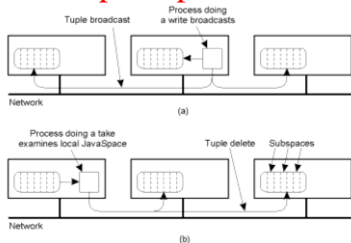
63

Synchronisation and replication

- Synchronisation
 - Blocking operations on tuples (read, take)
 - Transactions
- Replication
 - Replicated lookup service (service discovery)
- Fault tolerance
 - Reliable communication (RMI over TCP or HTTP)
 - Transactions

64

Simple replication

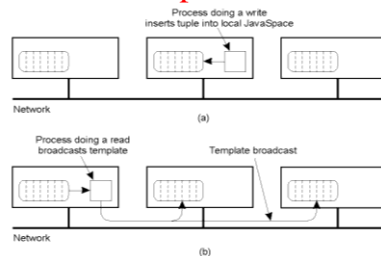


Assume broadcast cheap:

- a) tuples broadcast on write
- b) reads local but removing instance by take must be broadcast

65

No Replication

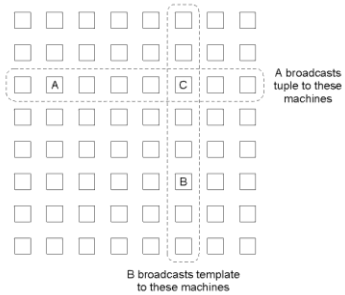


Primary-based protocol:

- a) write done locally.
- b) read or take requires template tuple to be broadcast

66

Quorum-Like Partial Replication



67

Jini Summary

Adaptation of Linda ideas to Java

- Separates communication from synchronization
 - support for loosely-coupled processes
 - e.g. to coordinate legacy non-distributed systems
- Performance challenge
 - scaling – no replication option scales very well
 - solution?
 - use coordination sparingly
 - use for long-term concerns (e.g. maintaining web caches)

Read 13.1, 13.2 (13.2.1-13.2.2), 13.6, 13.7.1

68