Distributed File Systems

CS 451 Lecture 2003

Prof. RK Joshi, CSE, IIT Bombay

What's a DFS?

- A distributed implementation of the classical file system
- To its clients, DFS should look like a conventional FS
  - Dispersion of servers and multiplicity of storage devices should be transparent (Ideally) to clients

What's a DFS?

- A distributed implementation of the classical file system
- To its clients, DFS should look like a conventional FS
  - Dispersion of servers and multiplicity of storage should be transparent (Ideally) to clients

Network transparency

- Implies that client uses the same set of FS abstractions –
  - No distinction is made between a remote and a local file
  - All internal handling is done by the DFS

User Mobility

- Example:
  - User can login from any machine
  - Home directory is made available at that machine at the same path

Performance Overheads

- Should be compatible to that of local file system
- User should not ‘see’ the difference

Fault Tolerance

- Communication failures, failures of servers, delays in storage media etc. should be tolerated to extent possible
- Graceful degradation – continue to function in a degraded form instead of crashing the service
  - Degradation could be of performance, functionality or both
  - i.e. not to halt the whole system when one or two components fail

Scalability

- Scalable system reacts more gracefully to increased load than a non-scalable system (A relative property)
  - i.e. reaches saturation later than a non-scalable system
  - Also performance degrades more moderately than a non-scalable system

Scalability Problems

- Adding new resources
  - May generate indirect load on existing resources
- Additions may need design modifications
- Related to fault tolerance

Naming & Transparency

- Naming = mapping between logical and physical objects
- Location transparency
  - The name of the file does not reveal the physical location (Locus, NFS, Sprite)
- Location independence
  - The name of the file need not be changed when the physical allocation changes (Andrews)
  - (file mobility/migration) –
  - stronger than location transparency
  - Dynamic mapping

Naming Scheme I

- Name by host name and local name
  - Host:local-name
    - Guarantees unique names
- The above is not location transparent
- It's not location independent
- But is network transparent (same set of calls for local and remote files)

Naming Scheme II

- Mount remote directories to local name spaces
- Once mounted, location transparent
- Shared namespace may not be identical on all machines (user mobility)
- If machine goes offline, directories become unavailable
- Control on permissions for attach/mount operation

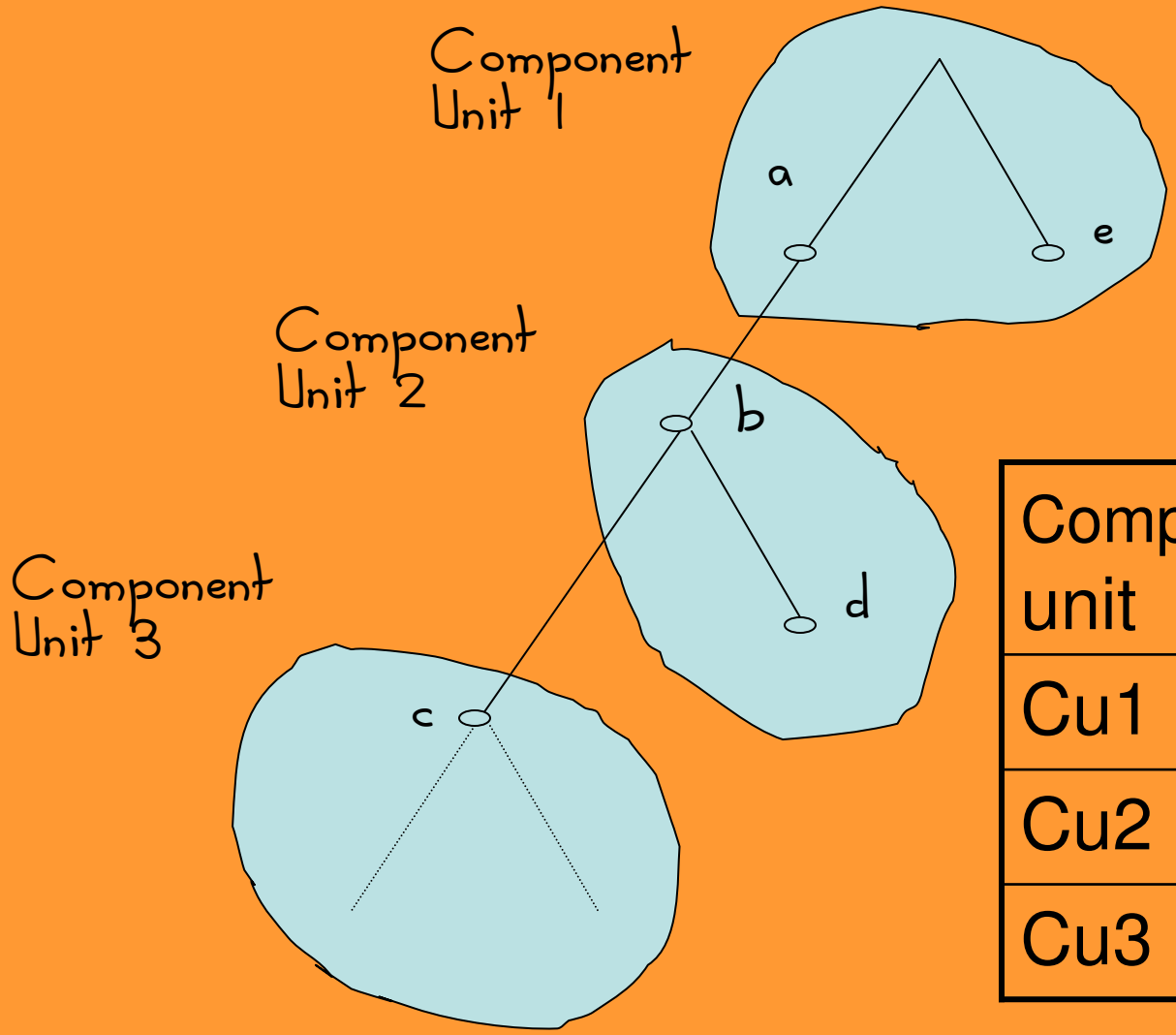
Naming Scheme III

- A single global logical name structure
- Same namespace is visible to all clients
- But local files (/dev, /proc, /tmp) make this goal difficult to attain

Pathname Translation

- Given path: /a/b/c
- How does a conventional fs translate this path to the actual location of the file?
  - Recursive lookup: i.e. lookup first in '/', file 'a' and then repeat the lookup procedure recursively on the remaining path, terminating when no path remains; the last result is returned.

Pathname Translation for naming scheme III



Location table

Component unit	Server location
Cu1	everest
Cu2	Voxel
Cu3	surya

Example: Pathname translation for /a/b/c initiated on everest

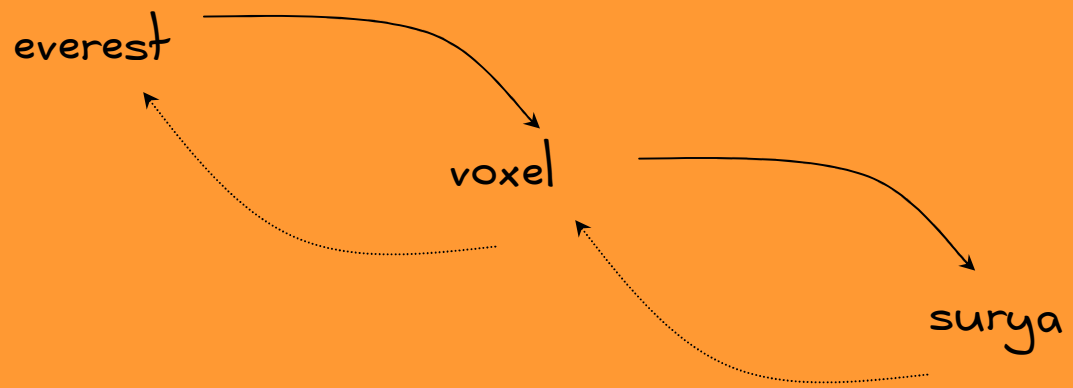
- location table available to all machines
- Start from /
- a is local
- Lookup for b: b is remote
- Pass on b/c to voxel
- b is local on voxel, c is remote
- Pass on c to surya
- c is found
- Low level id for /a/b/c is returned to client

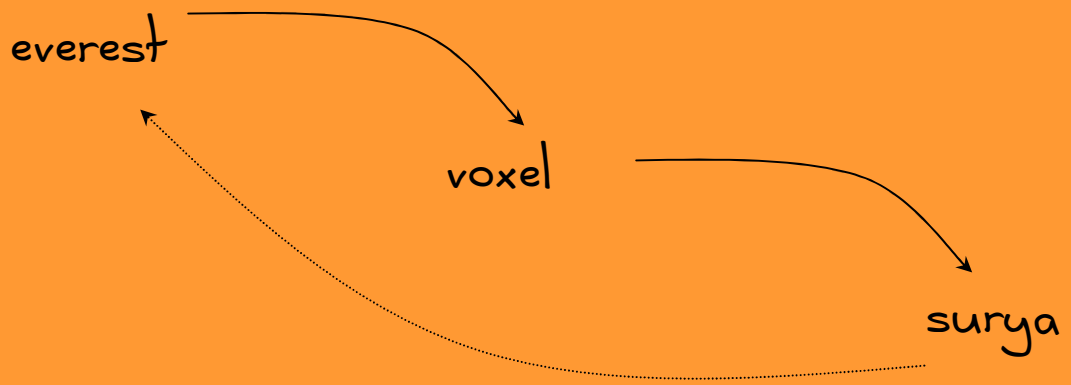
Variations in path lookup

- Does the machine return to the client
  - or
- Should it delegate recursively?

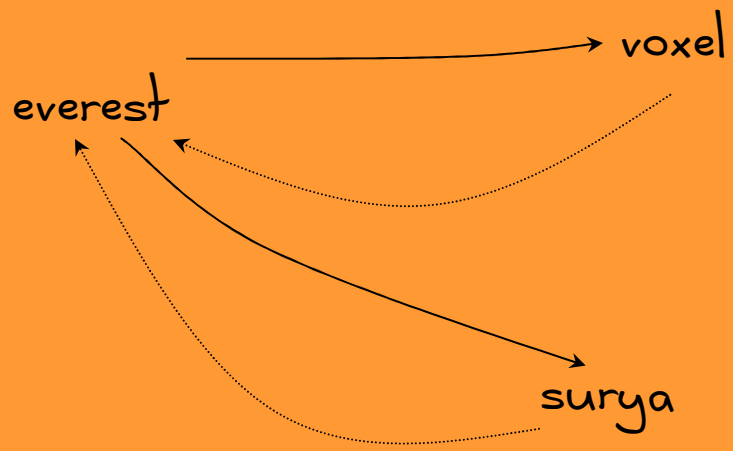
  should every request carry client identifier, or should a recursive call be returned to the recent caller?

Recursive Lookup (delegate)





Client iterates



To be continued in next lecture

- Visit this pdf again

Mapping of file names to location

- /a/b/c → cu3/11 which says, cu3 is the component unit for the file, and 11 is the idenitifier in that unit
- The mapping /a/b/c → <cu3, 11> is not invalidated upon migrating cu3 to another machine
- A second level mapping stores the actual location information on disk

Using ‘Hints’

- A hint is like a cached information
- But not exactly like a cache since it may also be incorrect
- Hint:
  - In case of incorrect information, there is no negative effect, but additional overheads
  - In case of correct information, lookup is faster
  - (used in Andrews file system)
  - If a hint is wrong, some systems do a broadcast of correct information

Semantics of Sharing

- What happens when 2 or more applications use the same file concurrently?
- Semantics with concurrent reads and writes?
  - » High level applications such as databases use their own mechanisms to control concurrency (e.g. locks)
  - » They don’t rely on FS semantics

Unix Semantics

- Every read sees the effects of all previous writes in a DFS
  - Writes by a client are visible to all clients who have that file open
  - Sharing of file pointer is possible
- Effects of file operations can be totally arbitrary as scheduling may determine the actual sequence

Session Semantics

- Write to open file are visible to local clients but invisible to remote clients who may have the file open simultaneously
- Once a file is closed, the changes are made visible to only later sessions
- i.e. each client/machine may have its own image

Which one is harder to achieve?

- Unix semantics or session semantics?
- Why?

Immutable shared files semantics

- Declare shared files as immutable
- These can now be opened by as many clients, but cannot be modified

Transaction like semantics

- Final effect is that of executing sessions in some serial order
- i.e. a file is r/w locked by sessions

Remote Access Method

- Remote service: for every access, use the remote service
- Caching
  - Cache consistency problem
    - Is the cached copy consistent with master copy?
  - Cache unit size?
    - Can you implement ***read-ahead****?*
  - Cache location?
    - On local disk? Or in local memory?
  - Cache Modification (dirty block flush) policy?
    - Affects system performance

Cache Modification policy

- Write-through
  - Reliable
    - when client, the writing process crashes, little info is lost
  - Equivalent to using remote service for write accesses: poor write performance
- Delayed-write
  - Delay updates to master copy
  - Write modifications to cache
  - If data is deleted before written back, update is saved
  - When to write?
    - When the block is about to be ejected from cache
    - Periodically (compromise between write-through and delayed write)
      - E.g. Unix uses 30 seconds delayed-write policy for flushing
    - Write-on-close: write data back to server when file is closed
      - Close operation gets delayed
      - Does not reduce n/w traffic for short files with fewer modifications
      - Useful for long sessions with frequent modifications
- Write-on-close: suitable for session semantics
- Write-through: suitable for unix semantics

Who performs Cache validation?

- Client initiated
  - client checks with the server whether local local data is consistent with master copy
  - check before every access
  - Check on first access to a file
  - Check periodically
- Server initiated
  - Server takes the responsibility
  - When server detects potential for inconsistency (e.g. caching by clients in conflicting modes)
    - Session semantics: on close, server can notify cache invalidation to other clients
    - Unix semantics: on write request from client, invalidate remote copies and switch to remote service access
  - Violates the client-server model

serializability

- Do the “caching-oriented” unix semantics and session semantics guarantee serializability?

Fault tolerance

- Stateful servers
  - Server maintains information about clients
- Stateless servers
  - Server does not remember anything about client after client finishes its single request

Recoverable and Robust files

- A file is recoverable if it's possible to revert the file to its earlier consistent state if an operation fails or gets aborted by the caller
- A file is robust if it is guaranteed to survive crashes of storage device and decays of storage medium

Available files

- A file is available if it can be accessed whenever needed despite machine, storage device crashes and communication failures

Readings

- Levy and Silberschatz, Distributed File Systems, ACM Computing Surveys, Dec. 1990