

Distributed File Systems

Examples

By Mads Pultz mpultz@diku.dk

Niels Boldt boldt@diku.dk

Outline

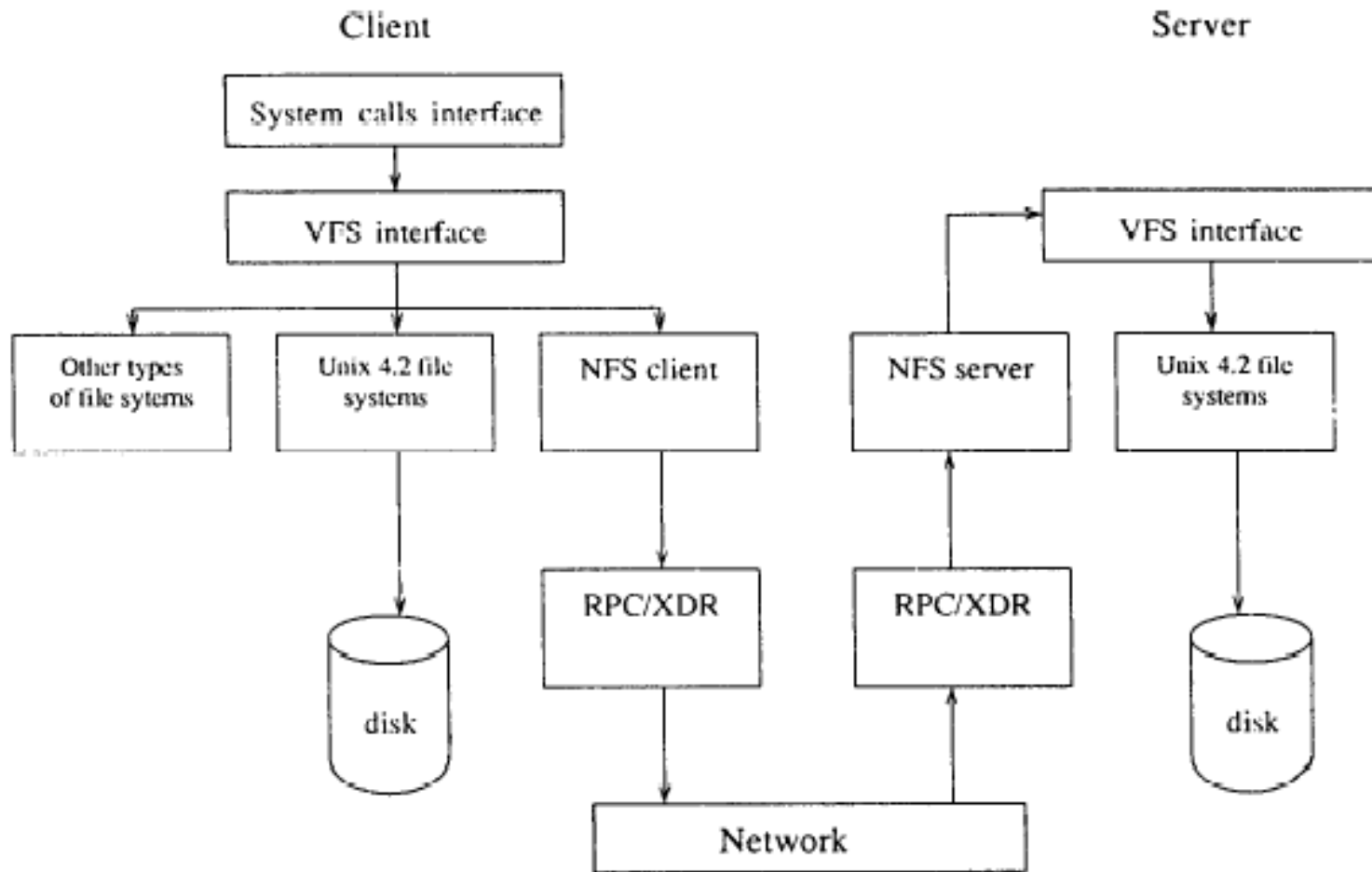
◆ Three examples

- NFS (Network File System)
- AFS (Andrew File System)
- Coda

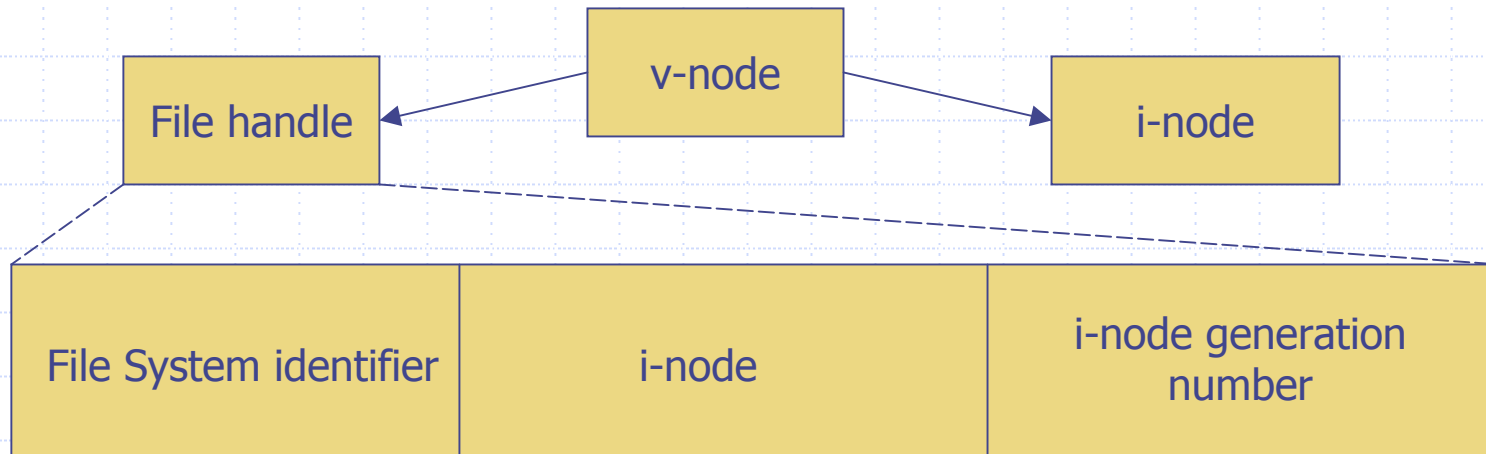
NFS

- ◆ A specification for a distributed file system (by Sun, 1984)
- ◆ Implemented on various OS's
- ◆ De facto standard in the UNIX community
- ◆ Latest version is 4 (2000, RFC3010)
- ◆ Client-server file system

NFS – overview



NFS – v-nodes



- ◆ v-node contains a reference to a file handle if the file is remote or an i-node if the file is local
- ◆ File system identifier
 - Unique number generated for each file system (in UNIX stored in super block)
- ◆ i-node and i-node generation number
 - UNIX file metadata

NFS – transparency

◆ Access transparency

- After mount API same as for UNIX

◆ Location transparency

- File names does not reveal anything about their locations (other than the mount points)

NFS – pathname translation (1)

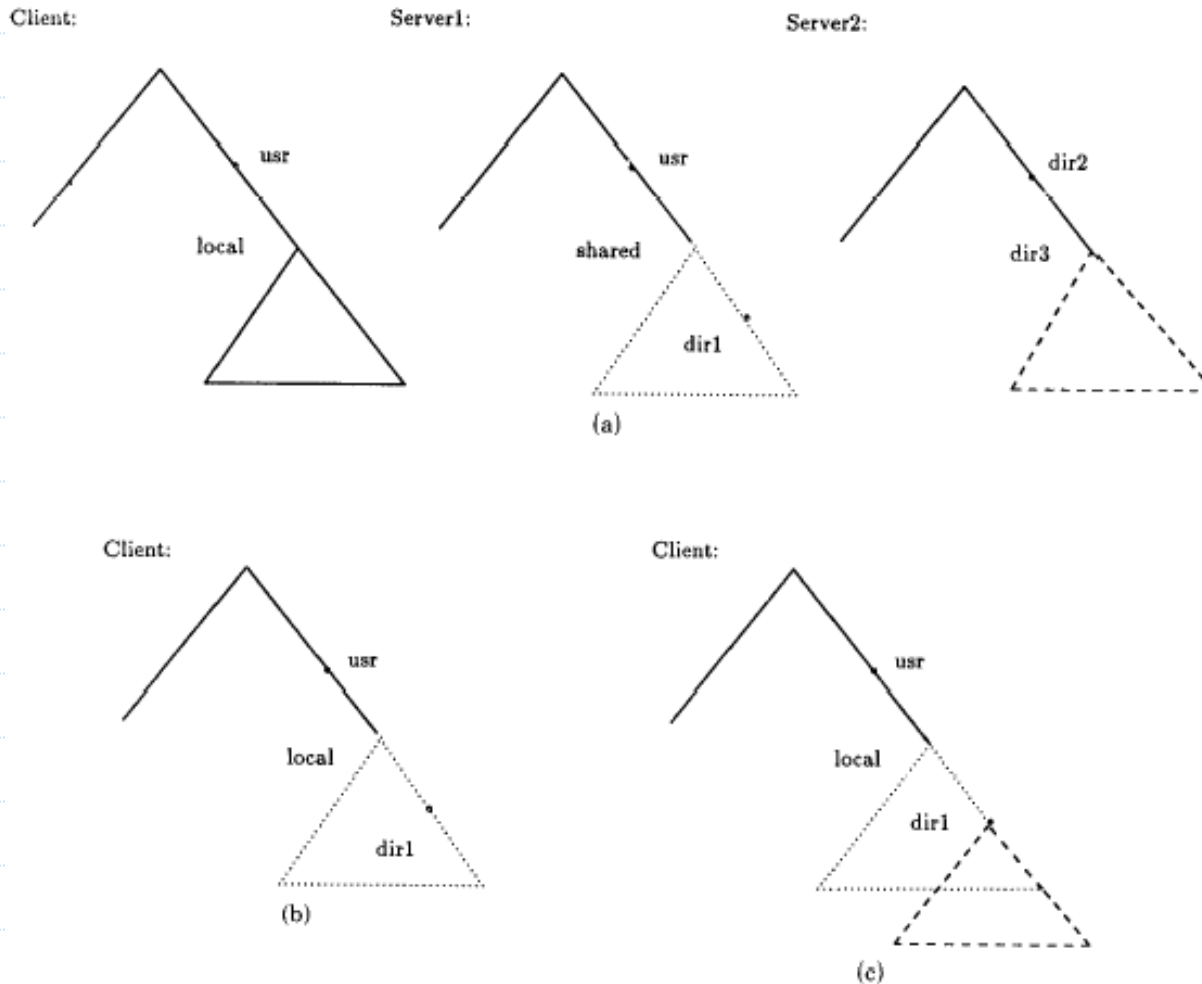


Figure 5. NFS joins independent file systems (a), by mounts (b), and cascading mounts (c).

NFS – pathname translation (2)

- ◆ Is done iteratively by client
- ◆ /usr/local/dir1/myfile
 - Lookup(/ I-node, usr) → /usr I-node
 - Lookup(/usr I-node, local) → /usr/local file handle
 - ◆ Server 1 is contacted
 - Lookup(/usr/local file handle, dir1) → /usr/local/dir1 file handle
 - ◆ Server 2 is contacted
 - Lookup(/usr/local/dir1 file handle, myfile) → /usr/local/dir1/myfile file handle
 - ◆ Server 2 is contacted
- ◆ Server 1 cannot lookup dir1 for client because dir1 is something else on server 1 than on client
- ◆ Lookups are cached

NFS – server caching

◆ Reads

- Uses the local file system cache (for example UNIX read-ahead)

◆ Writes

- Write-through (synchronously, no cache)
- Commit on close (standard behaviour in v3)

NFS – client caching (reads)

- ◆ Clients are responsible for validating cache entries (one of the reasons why the server is stateless)
- ◆ Timestamp system used
 - All timestamps are issued by server
- ◆ A cache entry is valid if one of the following are true:
 - Cache entry is less than t seconds old
 - Modified time at server is the same as modified time on client
- ◆ t is 3-30 s for files, 30-60 s for directories

NFS – client caching (writes)

◆ Delayed writes:

- Modified files are marked dirty and flushed to server on close (or sync)

◆ Bio-daemons (block input-output):

- Read-ahead requests are done asynchronously
- A write request is submitted when a block is filled

NFS – sharing semantics

- ◆ Not UNIX semantics
- ◆ Not real clear because of timing dependencies
- ◆ Consistency issues arise if multiple users are working on the same file
 - Example: Fritz and Henning have a file cached. Fritz opens the file and modifies it, then he closes the file. Henning then opens the file (before t seconds have elapsed) and modifies it as well. Then he closes the file. Are both Fritz' and Henning' modifications present in the file?
- ◆ Locking part of v4 (byte range, leasing)

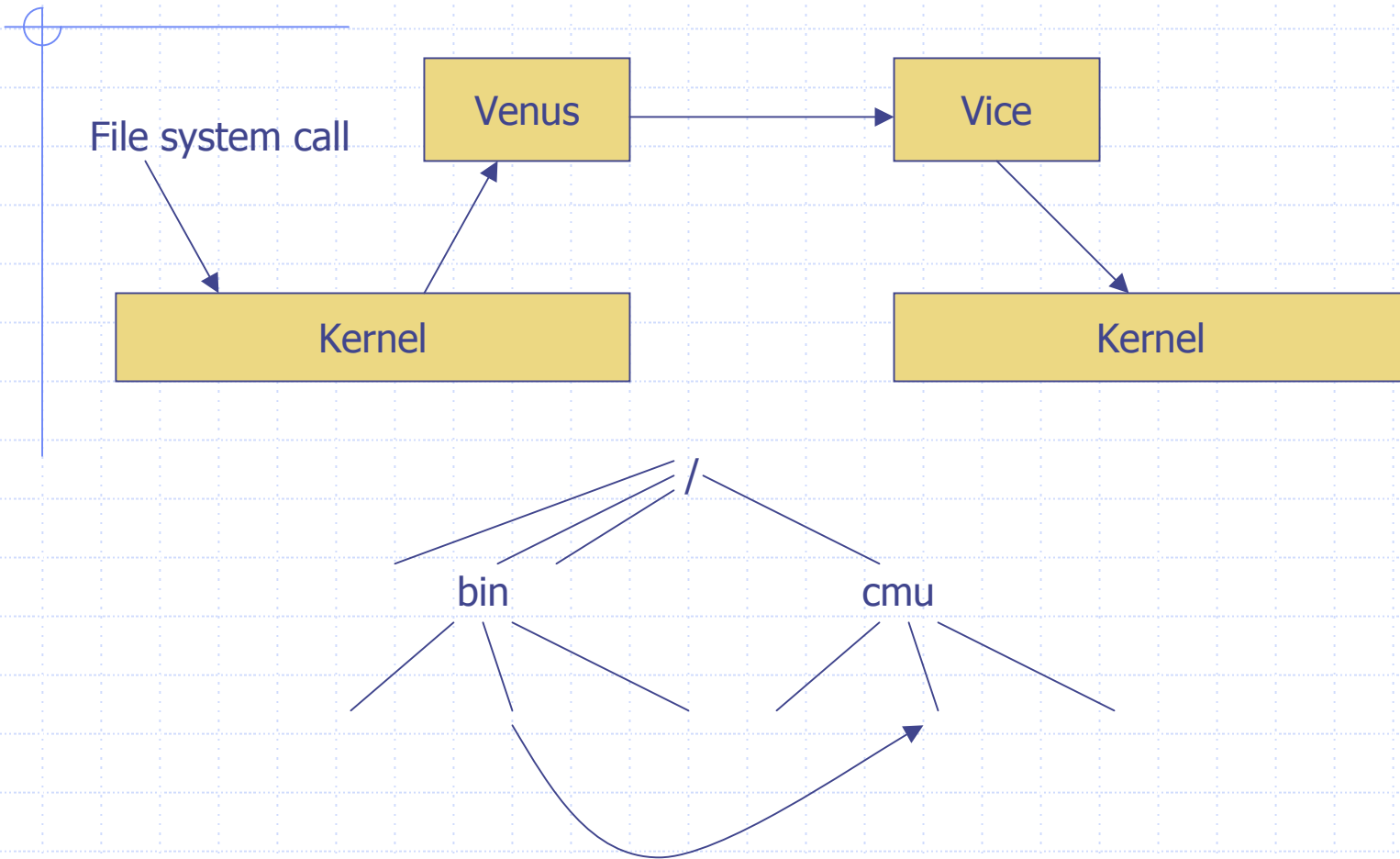
NFS – conclusion

- ◆ Client-server system
- ◆ Location transparency
- ◆ Server stateless
- ◆ Consistency issues
- ◆ Availability not a design goal
- ◆ Various implementations
- ◆ Replication of read-only files

AFS

- ◆ Developed at Carnegie Mellon University starting in 1984
- ◆ Design goal is scalability

AFS – overview



AFS - transparency

◆ Access transparency

- API same as for UNIX

◆ Location independency

- Global name space (/cmu)

- File identifier:

Volume number	File handle	Uniquifier
---------------	-------------	------------

- File /cmu/foo located at machine A can be moved to machine B
- A volume location map is replicated at each server

AFS – scalability

- ◆ Scalability is achieved through
 - Whole-file serving
 - ◆ Entire files are transmitted to clients (64 KB blocks)
 - Whole-file caching
 - ◆ Large disk cache
 - Clustering

AFS – caching

- ◆ Validation of cache entries is done locally
- ◆ Servers job to invalidate clients cache entries (makes server stateful)
- ◆ Invalidation is done through callbacks.
- ◆ Callbacks are initially set up for all files in client cache. If modification to a file occur from another client the server breaks the callback. If callback exist a cache entry is valid.

AFS – update semantics

- ◆ Session semantics
- ◆ Successful open
 - latest(F,S) or (lostCallback(S,T) and inCache(F) and latest(F,S,T))
 - lostCallback(S,T) = A callback from the server has been lost during the last T seconds
 - T is typically 10 minutes

AFS – conclusion

- ◆ Session semantics
- ◆ Scalable
- ◆ Location independency
- ◆ Stateful server
- ◆ Consistency issues
 - Two simultaneous writes to the same position in a file. Former write is not present after latter write.
- ◆ Replication of read-only files

Coda

- ◆ Descendent of AFS
- ◆ Developed at Carnegie Mellon University since 1987
- ◆ Primary design goal is constant data availability
 - Achieved through
 - ◆ Server replication
 - ◆ Disconnected operation

Coda – overview

- ◆ A volume is replicated at a set of servers (volume storage group, VSG)
- ◆ Each Venus process has access to a subset of VSG (available VSG, AVSG)
- ◆ Each Venus process has a preferred server in AVSG.

Coda – update semantics

- ◆ Weaker than AFS semantic (not session semantics)
- ◆ Successful open
 - $AVSG \leftrightarrow \emptyset$ and ($\text{latest}(F, AVSG)$ or ($\text{lostCallback}(AVSG, T)$ and $\text{inCache}(F)$ and $\text{latest}(F, AVSG, T)$)) or ($AVSG = \emptyset$ and $\text{incache}(F)$)
 - $\text{lostCallback}(AVSG, T)$ = A callback from the server has been lost during the last T seconds
 - T is typically 10 minutes

Coda – server replication

- ◆ Upon close modifications propagate to AVSG with multiRPC (multicast)
- ◆ All servers are contacted when opening a file to make sure the preferred server has the latest copy and that all replicas are in sync
- ◆ So, clients are responsible for server replication

Coda – disconnected operation

- ◆ Disconnected users can operate on files in their cache
- ◆ Can specify a list of files which Venus will try to keep in cache at all times
- ◆ Servers in VSG\AVSG are periodically polled
- ◆ Modified files are automatically transferred to preferred server upon reconnection

Coda - conflicts

- ◆ Coda Version Vectors (CVV) are used to detect conflicts
- ◆ Each file has a CVV associated with it
- ◆ Example: 3 servers and 2 clients
 - Two partitions $\{S1,S2,C1\}$ and $\{S3,C2\}$
 - Initially CVV is $(1,1,1)$
 - C1 updates file $\rightarrow (2,2,1)$
 - C2 updates file $\rightarrow (1,1,3)$
 - When the partitions merge we have a conflict because neither $CVV_{C1} \geq CVV_{C2}$ nor $CVV_{C2} \geq CVV_{C1}$
- ◆ Automatic conflict detection
 - Some directory conflicts can be automatically resolved

Questions

- ◆ Do you think access transparency is feasible in a distributed environment (hint: literature WWWK94)
- ◆ Describe caching strategies in NFS and AFS and discuss pros and cons
- ◆ How does NFS and AFS handle simultaneous users? What is the semantics of sharing?
- ◆ Which environments does NFS and AFS best fit?