

DISTRIBUTED SYSTEMS - III

Tevfik Koşar

University at Buffalo
November 29th, 2012

Distributed File Systems

- Distributed file system (DFS) - a distributed implementation of the classical time-sharing model of a file system, where multiple users share files and storage resources over a network
- A DFS manages set of dispersed storage devices
- Overall storage space managed by a DFS is composed of different, remotely located, smaller storage spaces
- There is usually a correspondence between constituent storage spaces and sets of files

DFS Interface

- DFS provides access to and manipulation of data stored at remote servers using **file system interfaces**
- What are the file system interfaces?
 - Open a file, check status on a file, close a file;
 - Read data from a file;
 - Write data to a file;
 - Lock a file or part of a file;
 - List files in a directory, delete a directory;
 - Delete a file, rename a file, add a symlink to a file;
 - i.e. POSIX interface

3

File System vs Block-Level Interface

- Data are organized in files, which in turn are organized in directories
- Compare these with disk-level access or “block” access interface: [Read/Write, LUN, block#]
- Key differences:
 - Implementation of the directory/file structure and semantics
 - Synchronization

4

Buzz Words: NAS vs SAN

	NAS	SAN
Access Methods	File access	Disk block access
Access Medium	Ethernet	Fiber Channel and Ethernet
Transport Protocol	Layer over TCP/IP	SCSI/FC and SCSI/IP
Efficiency	Less	More
Sharing and Access Control	Good	Poor
Integrity demands	Strong	Very strong
Clients	Workstations	Database servers

5

Why is DFS Useful?

- Data sharing of multiple users
- User mobility
- Data location transparency
- Data location independence
- Replications and increased availability
- Not all DFS are the same:
 - Local-area vs Wide area DFS
 - Fully Distributed FS vs DFS requiring central coordinator

6

Issues in Distributed File Systems

- Naming (global name space)
- Performance (Caching, data access)
- Reliability (replication, recovery)
- Consistency (when/how to update/synch?)
- Security (user privacy, access controls)

7

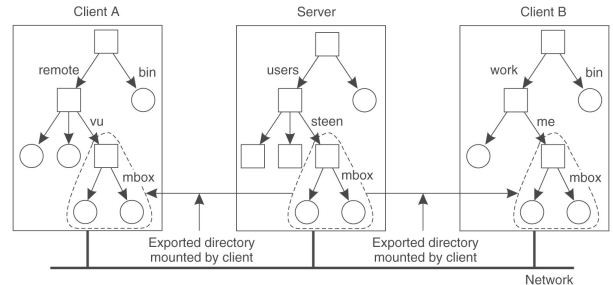
Naming of Distributed Files

- *Naming* – mapping between logical and physical objects.
- A *transparent* DFS hides the location where in the network the file is stored.
- **Location transparency** – file name does not reveal the file's physical storage location.
 - File name denotes a specific, hidden, set of physical disk blocks.
 - Convenient way to share data.
 - Could expose correspondence between component units and machines.
- **Location independence** – file name does not need to be changed when the file's physical storage location changes.
 - Better file abstraction.
 - Promotes sharing the storage space itself.
 - Separates the naming hierarchy from the storage-devices hierarchy.

DFS - Three Naming Schemes

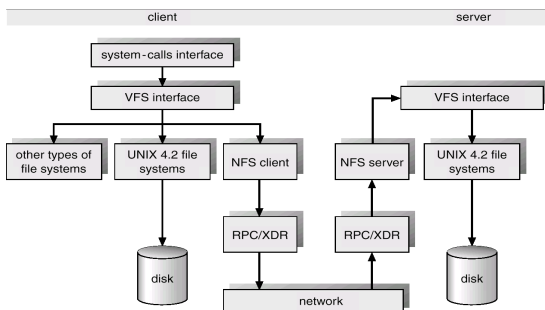
1. *Mount* remote directories to local directories, giving the appearance of a coherent local directory tree
 - *Mounted* remote directories can be accessed transparently.
 - Unix/Linux with NFS; Windows with mapped drives
2. Files named by combination of *host name* and *local name*;
 - Guarantees a unique system wide name
 - Windows *Network Places*, Apollo Domain
3. Total integration of component file systems.
 - A single global name structure spans all the files in the system.
 - If a server is unavailable, some arbitrary set of directories on different machines also becomes unavailable.
 - AFS

Mounting Remote Directories (NFS)



10

Mounting Remote Directories (NFS)



11

Mounting Remote Directories (NFS)

- Note:– *names* of files are not unique
 - As represented by *path names*
- E.g.,
 - Server A sees : /users/steen/mbox
 - Client A sees: /remote/vu/mbox
 - Client B sees: /work/me/mbox
- Consequence:– Cannot pass file “names” around haphazardly

12

DFS - Remote Service vs Caching

- *Remote Service* – all file actions implemented by server.
 - RPC functions
 - Use for small memory diskless machines
 - Particularly applicable if large amount of write activity
- *Cached System*
 - Many “remote” accesses handled efficiently by the local cache
 - Most served as fast as local ones.
 - Servers contacted only occasionally
 - Reduces server load and network traffic.
 - Enhances potential for scalability.
 - Reduces total network overhead

13

DFS Caching - File Access Performance

- Reduce network traffic by retaining recently accessed disk blocks in local *cache*
- Repeated accesses to the same information can be handled locally.
 - All accesses are performed on the cached copy.
- If needed data not already cached, copy of data brought from the server to the local cache.
 - Copies of parts of file may be scattered in different caches.
- *Cache-consistency* problem – keeping the cached copies consistent with the master file.
 - Especially on write operations

14

DFS - File Caches

- In client memory
 - Performance speed up; faster access
 - Good when local usage is transient
 - Enables diskless workstations
- On client disk
 - Good when local usage dominates (e.g., AFS)
 - Caches larger files
 - Helps protect clients from server crashes

15

DFS - Cache Update Policies

- When does the client update the master file?
 - I.e. when is cached data written from the cache to the file?
- *Write-through* – write data through to disk ASAP
 - I.e., following *write()* or *put()*, same as on local disks.
 - Reliable, but poor performance.
- *Delayed-write* – cache and then write to the server later.
 - Write operations complete quickly; some data may be overwritten in cache, saving needless network I/O.
 - Poor reliability
 - unwritten data may be lost when client machine crashes
 - Inconsistent data
 - Variation – scan cache at regular intervals and flush *dirty* blocks.

16

DFS - File Consistency

- Is locally cached copy of the data consistent with the master copy?
- *Client-initiated* approach
 - Client initiates a validity check with server.
 - Server verifies local data with the master copy
 - E.g., time stamps, etc.
- *Server-initiated* approach
 - Server records (parts of) files cached in each client.
 - When server detects a potential inconsistency, it reacts

17

DFS - File Server Semantics

- *Stateful Service*
 - Client *opens* a file (as in Unix & Windows).
 - Server fetches information about file from disk, stores in server memory,
 - Returns to client a *connection identifier* unique to client and open file.
 - Identifier used for subsequent accesses until session ends.
 - Server must reclaim space used by no longer active clients.
 - Increased performance; fewer disk accesses.
 - Server retains knowledge about file
 - E.g., read ahead next blocks for sequential access
 - E.g., file locking for managing writes
 - Windows

18

DFS - File Server Semantics

- *Stateless Service*
 - Avoids *state* information in server by making each request self-contained.
 - Each request identifies the file and position in the file.
 - No need to establish and terminate a connection by open and close operations.
 - Poor support for locking or synchronization among concurrent accesses

19

DFS - Server Semantics Comparison

- Failure Recovery: *Stateful server* loses all volatile state in a crash.
 - Restore state by recovery protocol based on a dialog with clients.
 - Server needs to be aware of crashed client processes
 - orphan detection and elimination.
- Failure Recovery: *Stateless server* failure and recovery are almost unnoticeable.
 - Newly restarted server responds to self-contained requests without difficulty.

20

DFS - Server Semantics Comparison

- Penalties for using the robust stateless service: –
 - longer request messages
 - slower request processing
- Some environments require stateful service.
 - Server-initiated cache validation cannot provide stateless service.
 - File locking (one writer, many readers).

21

DFS - Replication

- *Replicas* of the same file reside on failure-independent machines.
- Improves availability and can shorten service time.
- Naming scheme maps a replicated file name to a particular replica.
 - Existence of replicas should be invisible to higher levels.
 - Replicas must be distinguished from one another by different lower-level names.
- Updates
 - Replicas of a file denote the same logical entity
 - Update to any replica *must* be reflected on all other replicas.

22

Two Popular DFS

- NFS: Network File System (from SUN)
- AFS: the Andrew File System

23

AFS - NFS Quick Comparison

- NFS: per-client linkage
 - Server: export /root/fs1/
 - Client: mount server:/root/fs1 /fs1 → fhandle
- AFS: global name space
 - Name space is organized into Volumes
 - Global directory /afs;
 - /afs/cs.wisc.edu/vol1/...; /afs/cs.stanford.edu/vol1/...
 - Each file is identified as <vol_id, vnode#, vnode_gen>
 - All AFS servers keep a copy of “volume location database”, which is a table of vol_id → server_ip mappings

24

AFS - NFS Quick Comparison

- NFS: no transparency
 - If a directory is moved from one server to another, client must remount
- AFS: transparency
 - If a volume is moved from one server to another, only the volume location database on the servers needs to be updated
 - Implementation of volume migration
 - File lookup efficiency
- Are there other ways to provide location transparency?

25

More on NFS

- AFS is a **stateful** service
- NFS is a **stateless** service
- Server retains no knowledge of client
 - Server crashes invisible to client
- All hard work done on client side
- Every operation provides *file handle*
- Server caching
 - Performance only
 - Based on recent usage
- Client caching
 - Client checks validity of caches files
 - Client responsible for writing out caches

26

More on NFS

- No locking! No synchronization!
- *Unix file semantics* not guaranteed
 - E.g., *read* after *write*
- *Session semantics* not even guaranteed
 - E.g., *open* after *close*
- Solution: *global lock manager*
 - Separate from NFS

27

NSF Failure Recovery

- Server crashes are transparent to client
 - Each client request contains all information
 - Server can re-fetch from disk if not in its caches
 - Client retransmits request if interrupted by crash
 - (i.e., no response)
- Client crashes are transparent to server
 - Server maintains no record of which client(s) have cached files.

28

DFS Summary

- *Performance* is always an issue
 - Tradeoff between performance and the semantics of file operations (especially for shared files).
- *Caching* of file blocks is crucial in any file system, distributed or otherwise.
 - As memories get larger, most read requests can be serviced out of file buffer cache (local memory).
 - Maintaining coherency of those caches is a crucial design issue.
- Current research addressing disconnected file operation for mobile computers.

29

Any Questions?



21

Acknowledgements

- “Operating Systems Concepts” book and supplementary material by A. Silberschatz, P. Galvin and G. Gagne
- “Operating Systems: Internals and Design Principles” book and supplementary material by W. Stallings
- “Modern Operating Systems” book and supplementary material by A. Tanenbaum
- R. Doursat and M. Yuksel from UNR