Chapter 17 - Distributed File Systems (DFS)

1. Definitions

- **DFS** is a file system where multiple users share files & directories which are physically dispersed among different distributed systems, using network and low-level protocols. A DFS system consists of:
  - **Service**: is the certain type of functions provided by the software entity, running on the systems, to the client.
  - **Server**: is the computer system that consists of multiple services.
  - **Client**: is the process that invokes and consumes the service from the server.
    - *Client Interface for a file service is the form of primitive file operations (Create/Delete/Read/Write), should be transparent. Client Interface does not distinguish between local and remote files.* [1] Ans: … The client **needs to know about** the protocol & structure & which server to connect to on the server.
- **Component Unit**: smallest set of files that can be stored on a system
- **DFS Performance Measurement**: 2 factors, (1) time to satisfy the Service Request, (2) Transparency *(how comparable the DFS is to conventional file system)*

2. Naming & Transparency

Naming is how the DFS maps a logical file/directory to a remote physical storage medium (multilevel mapping). A **transparent** DFS hides the physical location of the file, which can be replicated on multiple remote systems. Base on client's proximity to the server, the location of the nearest file replicas are returned.

2.1. Naming Structures

There are 2 types: **Location Transparency** – *i.e. one can't determinte the server hosting the file by the filename*, and **Location Independence** – *filename can stay intact when file physical location is changed*.

Comparison between Location Transparency and Location Independence

Location Transparency	Location Independence
Simple, and more desirable for security (as user does not know where the file location is)	*Convenient as user does not need to lookup where the file locates even it's moved between physical disk drives.*
Static, does not support file migration	*Dynamic, support file migration*
Movement of file names & disk are manual	*Migration is done by software*
not popular	*more popular (implemented in AFS - Andrew File System)*
Convenient to share remote files as similar as local files	*better abstraction since logical data container not attached to specific location*
Able to balance the utilization of disks across the system	*Naming hierarchy separated from storage devices hierarchy*

> In current trend, **OS and Networking** software are locally stored, **User Data and System utilities** are remotely stored.

2.2. Naming Schemes

1. Filename = `hostname` + `local name`
2. Attach remote directories to local mount points. Used in Sun Open Network Computing NFS
3. Total integration of Componenet File Systems

2.3. Implementation Techniques

2.3.1. Hierarchy

- Group files into a Component Unit, and map the component unit.
- Filename (Lower-Level file identifier) indicates the Component Unit.

- then Secondary-Level Component Unit indicates the Locations.

  Example: AFS uses Component Units, Lower level file identifier

2.3.2. Remote Service

Uses RPC for each request, and caching to reduce traffic and I/O.

2.3.3. Cache

Cache Scheme: only blocks, not entire file are copied from server to local client. LRU is used to contain cache size. Since copies of master file are scattered on different systems, there will be **Cache-Consistency Problem** *(keeping cached copies consistent with the master file)*.

Cache Size: Cache can be large, or small. **Large caches** increase hit and miss ratio at the same time, have more data transferred) and increase consistency problems. **Small caches** are not useful with large block size. *AFS cache size is 64KB, Unix block sizes are 4KB and 8KB.*

Cache Location: either in memory or on disk. **Disk caches** is more reliable and does not require re-fetch after recovery. **Main-memory caches** is faster, performance speedup in larger memory, workstation can go diskless. [2]

Ans: Memory cache is more reliable than disk cache (false statement).

> **Most DFSs only implement memory cache**, not disk caching.
> **NFS**: use both Caching (main-memory) and RPC. However in Solaris 2.6+, NFS uses Memory, Disk cache, then RPC in corresponding order.

Cache Update Policy: policy to update master copy.

- **Write-through**: immediately write to disk as soon as updates on cache. Reliable, but poor performance *(every update triggers a write to server, and network latency can cause waiting)*. a.k.a. Read Cache - Write RPC. [3]

  Ans: … it has poor performance in Read. (false statement)
- **Delayed-write**: update cache first, then update master later. Good performance, fast turnaround, only last update is going to master, but poor reliability *(lost unwritten data if user machine crashes)*.
  - Mechanism: Scan cache at regular intervals and flush the variation (difference since last scan) of modified blocks or removed blocks are flush to server, result in good performance and client cache might not be flushed for a long time.
  - Variation **Write-on-close**: update master when the file is closed, good for frequently modified files or files open for long periods (less refresh, less hit on the traffic).

> - NFS uses delay-write, syncs metadata changes synchronously to the server to avoid directory structure corruption if Client or Server crashes.
> - AFS uses Write-on-close, delay closing process on client until file is written on server.

2.3.4. Consistency

Main question about consistency: is Local cached data consistent with Master Copy?
There are 2 solutions:

- **Client initiated approach**: cache validity check is initiated by the Client *(session base, thus stateful)*. It checks *(a) before every access, or (b) on first access, or © at fixed interval time*.
- **Server initiated approach**: cache validity check is initiated by the Server *(stateful as well)*. It records and resolve conflicts if it detects a potential inconsistency. Caching could be disabled for particular file while switching to *Remote Server mode*.

2.3.5. Comparisons

	Caching	**Remote Service**
Pros	Resource access faster with local cache	Simplicity
	lesser traffic & load, especially with large data blocks	Suitable for diskless, small-memory-capacity systems
	Suitable for **Stateful** connection	Suitable for **Stateless** connection
Cons	Complicated, not so effecient when Writes are frequent than Read.	Slow, high network turnaround time *(as each request need to travel across the network)* and server operation overhead (each request requires server to open, read, return, close the file)

	Stateful	Stateless
	Server tracks each file being accessed by client	no knowledge about how file are used
	File identifier is used through the session, memory space (for caching on server side) is reclaimed when session ends or inactive	Each request is self-contained
	Prediction of file access & loading the content ahead	No requirement to keep open files list in memory. Take longer request time, slower processing, using low level naming scheme, and may serve duplicate request.
	May lose all volatile state if crash	almost unnoticeable after failure recovery

> NFS is Stateless File Service

2.3.6. File Replication

> Replication is useful redundancy for improving availability.

2.4. AFS

- Has 2 namespaces: local, and shared.
  - Local: its root file system
  - **Shared**: made up of several component units (volumes), communicate to server through Vice, using fid to identifies the Vice. A fid contains:
    - Volume Number: access point to server, change if server change. *(there is mapping of VolumeID to IP)*
    - Vnode Number: inodes of files in single volume.
    - Uniquifier: unique identifier, allows reuse of vnode numbers for same file path.
- Fid is location transparent, file moved on server does not affect fid. The Volume location is kept on Vice server thus Fid does not need to care about real location of the server.
- **Fundamental Architecture**: Use **whole file caching**, 64KB cache size. [4]
  - Open: Read cache from Vice
  - Close: write changes to Server.
  - Cache is assumed valid unless notified by Server using **callback policy** *(consistency of files are managed by the server, using callback for notification of changes)*. > **callback_policy**: when a file is cached, the server makes a note of this and promises to inform the client if the file is updated by someone else. Callbacks are discarded and must be re-established after any client, server, or

network failure, including a time-out. Re-establishing a callback involves a status check and does not require re-reading the file itself.

Ans: Whole File Caching.

- Security:
    - No execution of client program on Vice System.
    - Encrypted messages for communication
- Protection: has Access Lists for access control.
- **Implementation**: at UNIX kernel with a set of system calls, include
    - path-name translation (volume, file, fid)
    - Venus uses client’s disk for data caching. For file status caching, it uses virtual memory.
    - the Venus (AFS interface) and server use inodes for file access instead of path name.
    - Uses LRU algorithm for cache size containment.

2.5. NFS V4

- stateful, no mount protocol
- support `open()` and `close()` , Client local cache, file locks (by client to server)

Footnotes:

---

1. **False statement about DFS?** ↩
2. **What is wrong for memory cache?** ↩
3. **What is FALSE about Write-Through Cache Update Policy?** ↩
4. **What is the technique used by AFS?** ↩