Computer Networks (CS425)

Instructor: Dr. Dheeraj Sanghi

Prev | Next | Index

Network File System (NFS)

Network File System (NFS) is a distributed file system (DFS) developed by Sun Microsystems. This allows directory structures to be spread over the networked computing systems.

A DFS is a file system whose clients, servers and storage devices are dispersed among the machines of distributed system. A file system provides a set of file operations like read, write, open, close, delete etc. which forms the file services. The clients are provided with these file services. The basic features of DFS are multiplicity and autonomy of clients and servers.

NFS follows the directory structure almost same as that in non-NFS system but there are some differences between them with respect to:

- Naming
- Path Names
- Semantics

Naming

Naming is a mapping between logical and physical objects. For example, users refers to a file by a textual name, but it is mapped to disk blocks. There are two notions regarding name mapping used in DFS.

- **Location Transparency:** The name of a file does not give any hint of file's physical storage location.
- **Location Independence:** The name of a file does not need to be changed when file's physical storage location changes.

A location independent naming scheme is basically a dynamic mapping. NFS does not support location independency.

There are three major naming schemes used in DFS. In the simplest approach, files are named by some combination of machine or host name and the path name. This naming scheme is neither location independent nor location transparent. This may be used in server side. Second approach is to attach or mount the remote directories to the local directories. This gives an appearance of a coherent directory. This scheme is used by NFS. Early NFS allowed only previously mounted remote directories. But with the advent of automount , remote directories are mounted on demand based on the table of

mount points and file structure names. This has other advantages like the file-mount table size is much smaller and for each mount point, we can specify many servers. The third approach of naming is to use name space which is identical to all machines. In practice, there are many special files that make this approach difficult to implement.

Mounting

The mount protocol is used to establish the initial logical connection between a server and a client. A mount operation includes the name of the remote directory to be mounted and the name of the server machine storing it. The server maintains an export list which specifies local file system that it exports for mounting along with the permitted machine names. Unix uses /etc/exports for this purpose. Since, the list has a maximum length, NFS is limited in scalabilty. Any directory within an exported file system can be mounted remotely on a machine. When the server receives a mount request, it returns a file handle to the client. File handle is basically a data-structure of length 32 bytes. It serves as the key for further access to files within the mounted system. In Unix term, the file handle consists of a file system identifier that is stored in super block and an inode number to identify the exact mounted directory within the exported file system. In NFS, one new field is added in inode that is called the generic number.

Mount can be is of three types -

1. **Soft mount:** A time bound is there.
2. **Hard mount:** No time bound.
3. **Automount:** Mount operation done on demand.

NFS Protocol and Remote Operations

The NFS protocol provides a set of RPCs for remote operations like **lookup, create, rename, getattr, setattr, read, write, remove, mkdir** etc. The procedures can be invoked only after a file handle for the remotely mounted directory has been esta- blished. NFS servers are stateless servers. A stateless file server avoids to keep state informations by making each request self-contained. That is, each request iden- tifies the file and the position of the file in full. So, the server needs not to store file pointer. Moreover, it needs not to establish or terminate a connection by opening a file or closing a file, repectively. For reading a directory, NFS does not use any file pointer, it uses a **magic cookie**.

Except the opening and closing a file, there is almost one-to-one mapping between Unix system calls for file operations and the NFS protocol RPCs. A remote file operation can be translated directly to the corresponding RPC. Though conceptu- ally, NFS adheres to the remote service paradigm, in practice, it uses buffering and caching. File blocks and attributes are fetched by RPCs and cached locally. Future remote operations use the cached data, subject to consistency constraints.

Since, NFS runs on RPC and RPC runs on UDP/IP which is unreliable, operations should be idempotent.

Cache Update Policy

The policy used to write modified data blocks to the server's master copy has critical effect on the system performance and reliability. The simplest policy is to **write through** the disk as soon as they are placed on any cache. It's advantageous because it ensures the reliability but it gives poor performance. In server site this policy is often followed. Another policy is **delayed write**. It does not ensure reliability. Client sites can use this policy. Another policy is **write-on-close**. It is a variation of delayed write. This is used by Andrews File System (AFS).

In NFS, clients use delayed write. But they don't free delayed written block until the server confirms that the data have been written on disk. So, here, Unix semantics are not preserved. NFS does not handle client crash recovery like Unix. Since, servers in NFS are stateless, there is no need to handle server crash recovery also.

Time Skew

Because of differences of time at server and client, this problem occures. This may lead to problems in performing some operations like " make ".

Performance Issues

To increase the reliability and system performance, the following things are generally done.

1. Cache, file blocks and directory informations are maintained.
2. All attributes of file / directory are cached. These stay 3 sec. for files and 30 sec. for directory.
3. For large caches, bigger block size (8K) is benificial.

This is a brief description of NFS version 2. NFS version 3 has already been come out and this new version is an enhancement of the previous version. It removes many of the difficulties and drawbacks of NFS 2.

Andrews File System (AFS)

AFS is a distributed file system, with scalability as a major goal. Its efficiency can be attributed to the following practical assumptions (as also seen in UNIX file system):

- Files are small (i.e. entire file can be cached)
- Frequency of reads much more than those of writes
- Sequential access common
- Files are not shared (i.e. read and written by only one user)

- Shared files are usually not written
- Disk space is plentiful

AFS distinguishes between client machines (workstations) and dedicated server machines. Caching files in the client side cache reduces computation at the server side, thus enhancing performance. However, the problem of sharing files arises. To solve this, all clients with copies of a file being modified by another client are not informed the moment the client makes changes. That client thus updates its copy, and the changes are reflected in the distributed file system only after the client closes the file. Various terms related to this concept in AFS are:

- **Whole File Serving**: The entire file is transferred in one go, limited only by the maximum size UDP/IP supports
- **Whole File Caching**: The entire file is cached in the local machine cache, reducing file-open latency, and frequent read/write requests to the server
- **Write On Close**: Writes are propagated to the server side copy only when the client closes the local copy of the file

In AFS, the server keeps track of which files are opened by which clients (as was not in the case of NFS). In other words, AFS has **stateful servers**, whereas NFS has **stateless servers**. Another difference between the two file systems is that AFS provides **location independence** (the physical storage location of the file can be changed, without having to change the path of the file, etc.) as well as **location transparency** (the file name does not hint at its physical storage location). But as was seen in the last lecture, NFS provides only location transparency. Stateful servers in AFS allow the server to inform all clients with open files about any updates made to that file by another client, through what is known as a **callback**. Callbacks to all clients with a copy of that file is ensured as a **callback promise** is issued by the server to a client when it requests for a copy of a file.

The key software components in AFS are:

- **Vice**: The server side process that resides on top of the unix kernel, providing shared file services to each client
- **Venus**: The client side cache manager which acts as an interface between the application program and the Vice

All the files in AFS are distributed among the servers. The set of files in one server is referred to as a **volume**. In case a request can not be satisfied from this set of files, the vice server informs the client where it can find the required file.

The basic file operations can be described more completely as:

- Open a file: Venus traps application generated file open system calls, and checks whether it can be serviced locally (i.e. a copy of the file already exists in the cache) before requesting Vice for it. It then returns

a file descriptor to the calling application. Vice, along with a copy of the file, transfers a callback promise, when Venus requests for a file.
- Read and Write: Reads/Writes are done from/to the cached copy.
- Close a file: Venus traps file close system calls and closes the cached copy of the file. If the file had been updated, it informs the Vice server which then replaces its copy with the updated one, as well as issues callbacks to all clients holding callback promises on this file. On receiving a callback, the client discards its copy, and works on this fresh copy.

The server wishes to maintain its states at all times, so that no information is lost due to crashes. This is ensured by the Vice which writes the states to the disk. When the server comes up again, it also informs all the servers about its crash, so that information about updates may be passed to it.

A client may issue an open immediately after it issued a close (this may happen if it has recovered from a crash very quickly). It will wish to work on the same copy. For this reason, Venus waits a while (depending on the cache capacity) before discarding copies of closed files. In case the application had not updated the copy before it closed it, it may continue to work on the same copy. However, if the copy had been updated, and the client issued a file open after a certain time interval (say 30 seconds), it will have to ask the server the last modification time, and accordingly, request for a new copy. For this, the clocks will have to be synchronized.

---

back to top
Prev| Next | Index