CHAPTER 6

Module 6

Distributed File Systems and Name Service

Syllabus

Introduction and features of DFS, File models, File Accessing models, File-Caching Schemes, File Replication, Case Study: Distributed File Systems (DFS), Network File System (NFS), Andrew File System (AFS), Introduction to Name services and Domain Name System, Directory Services, Case Study: The Global Name Service, The X.500 Directory Service , Designing Distributed Systems: Google Case Study.

6.1 Introduction

- **File system** describes how files are structured, named, accessed, used, protected and implemented. Files are used for permanent use of information on secondary storage. It also provides sharing of information.
- In distributed system, files are available on several computers. Computers in distributed system can share these physically dispersed files by using distributed file system. Service offers particular function to client and it is a software entity running on some machines. Server runs service software on single machine. Client process invokes the service through some set of defined operations called as client interface.
- File system offers file services to clients. The set of primitive file operations are create a file, delete a file, read from a file, and write to a file. Client interface is formed with set of these operations. File server controls local secondary-storage devices such as disks on which files are stored. These files are accessed from these devices as per request of client.
- There can be different implementation for distributed file system (DFS). Server may run on dedicated machines. In other implementation both client and server may run on same machine. DFS can be part of distributed operating system or it can be a software layer managing communication between file system and network operating system. DFS should come into view to its client as conventional centralized file system. The servers and storage devices which are on different machines in network should be invisible to clients. DFS should fulfil the request of client by arranging the required files or data.
- As data transfer is involved in operation of DFS, its performance is measured with amount of time required to service the client request. Storage space managed by a DFS includes different and remotely located small storage spaces.
- DFS supports the following :
  - **Remote Information sharing :** Any node in the system can transparently access file irrespective of its location.
  - **User mobility :** DFS allows the user to work on different nodes at different times without physically relocating the secondary storage devices.
  - **Availability :** DFS keeps multiple copies of file on different nodes. Failure of any node or copy does not affect the operation.

- **Diskless workstations :** DFS provides transparent file accessing capability; hence, economical diskless workstations can be used.

- DFS provides following three types of services.
  - **Storage service :** DFS deals with storage and management of space on secondary storage device that is used to store files in file system. Files can be stored in blocks. It offers logical view of storage system by allowing operations for storing and accessing the data in them.
  - **True File service :** True file service supports operations on individual file. These are creating and deleting files, modifying and accessing data from files. Typical design issues to support these operations include file accessing techniques, file sharing semantics, file caching and replication mechanism, data consistency, multiple copy update protocol and access control mechanism.
  - **Name service :** Name service offers mapping between file names to file IDs. It is difficult for human to remember file IDs. Most file system uses directories to carry out mapping called as directory service. This service supports creation and deletion of directories, adding new file to directory, deleting the existing file from directory, changing name of the file etc.

6.2 Desirable Features of a Good Distributed File System

Following are the features of good DFS :

1. Transparency

- **Structure transparency:** DFS uses multiple file servers. Each file server is user or kernel process to control secondary storage devices of that node on which it runs. Client should be unaware about location and number of file servers and storages devices. DFS should treat the client just like single conventional file system offered by centralized time sharing OS.
- **Access Transparency:** Accessing the local and remote file should be carried out in similar manner. DFS should automatically locate accessed file and support to transporting the data to client.
- **Naming Transparency:** name of file should remain same when it is moved from one node to other. Its name should be location independent.
- **Replication Transparency:** Existence of multiple replicated copies and their location should remain hidden from clients.

2. User mobility

DFS should permit the user to work on different nodes at different times. Performance should not affect if user works on node other than his node. User's home directory should be automatically made available when user logs in on new node.

3. Performance

DFS must give performance same as centralized file system. User should not feel the need to place file explicitly to improve performance.

4. Simplicity and ease of use

User interface to file system should be simple and small number of commands should be supported. It should support for all types of applications. The semantics of DFS should be same as single conventional file system for centralized time sharing system.

5. **Scalability**

The DFS should also supports for growth of nodes and users in network. Such growth should not lead to service loss or performance degradation of the system.

6. **High Availability**

DFS should continue to function if partial failure occurs in one or more components. This failure can be communication link failure, node failure, and secondary storage failure. Degradation in performance due to failure must be in proportional to that failure.

7. **High Reliability**

DFS should support for reliability. It should automatically create backup copies of the critical files that can be loss in the failure. Users should ensure that there will be no or minimum loss of their data.

8. **Data Integrity**

Simultaneous accesses to shared file by many users should guarantee the integrity of data stored in it. These access requests to file from many users should be properly synchronized by using proper concurrency control mechanism.

9. **Security**

DFS should be secured so that it can offer the privacy to user's data. It should implement the security mechanism to protect file data from unauthorized access. Secured access right allocation policy should also be supported.

10. **Heterogeneity**

In large distributed system, machine's architecture and platforms used are different. DFS should support for heterogeneity so that users can use different computer platforms for different applications. DFS should allow to integrate a new type of node or storage area in simple manner.

6.3 File Models

Following are two file models based on structure and modifiability :

6.3.1 Unstructured and Structured Files

- **Unstructured Files :** File is unstructured sequence of data. Structure of the file appears to file server as un-interpreted sequence of bytes. The interpretation of structure and meaning of data in file is up to the program and OS is not interested in the same. This file model is used by MS-DOS and UNIX. Most of the operating system used this model as sharing of files by different applications is easier. Different application can view content of the files in different way.
- **Structured Files :** In this model, file is presented to file server as order sequence of records. Record is smallest unit of data that can be accessed. Different files in the same file system may have different size of records. To access the record in structure files with non-indexed records, its position within needs to be specified. For example seventh record from beginning of file. In structure files with indexed records, a record is accessed by using key value.
- File is identified by its name which is string of the characters. Every file has a name and its data. All operating systems associate other information with each file, for example, the date and time of file creation and the size of the file. Such extra information associated by operating system is called as attributes.

- The lists of attributes are not same for all the systems and vary from one operating system to another. No single existing system supports all of these attributes, but each one is present in some system.

6.3.2 Mutable and Immutable Files

- **Mutable Files :** Most of the OS uses this model. In this file, each update operation on file update overwrites its old content and new content is produced. Hence, file is represented as single stored sequence that is changed by each update operation.
- **Immutable Files :** In this file model, each update operation creates new version of the file. Changes are made in new version and old version is retained. Hence, more storage space is required.

6.4 File-Accessing Models

6.4.1 Accessing Remote Files

DFS may use one of the following models to service request of the client to access the file.

- **Remote Service Model :** In this model suppose client forward the request to server to access remote file. Naming scheme locates the server and actual data transfer between client and server is achieved through remote-service mechanism. In this mechanism, request for accesses is forwarded to server, which then performs accesses and returns the result to user or client. This is similar to disk accesses in conventional file system. Hence, data packing and communication overhead is significant in this model.
- **Data-Caching Model :** Caching can be used to improve performance of remote-service mechanism. In conventional file system, caching is used to reduce disk I/O. The main goal behind caching in remote-service mechanism is to reduce network traffic and disk I/O. If data is not available locally then it is copied from server machine to client machine and cached there. This cached data is then used by client at client side to process its request. This model offers better performance and scalability.
- All the future repeated accesses to this recently cached data can be carried out locally. This will reduce additional network traffic. Least Recently Used (LRU) algorithm can be used for replacing the cached data. Master copy of file is available on server and its part is scattered on many client machines. If copy of the file at client side modifies then its master copy at server should be updated accordingly called as cache-consistency problem.
- DFS caching is network virtual memory which works similar to demand-paged virtual memory having remote server as backing store. In DFS, data cached at client side can be disk blocks or it can be entire file. Actually more data are cached by client than actually needed so that most of the operations are carried out locally.

6.4.2 Unit of Data Transfer

The unit of data transfer is fraction of data that is transferred between client and server due to single read or write operation. In data-caching model of accessing remote files, following four models are used to transfer the data.

File-level Transfer Model

- In this model, whole file is transferred in either direction across the network. Hence, the network protocol overhead is required only once. Its scalability is better as it requires smaller number of accesses to serve and hence, reduces server load and network traffic.
- Disk access routines on server always better optimized as it is known that accesses are for file instead of random blocks.
- One time transfer is required to transfer the file to the form compatible to client file system. The drawback of this form of transfer is that it requires sufficient space at client as entire file is cached. Amoeba, CFS and Andrew File System uses this transfer model.

Block-level Transfer Model

- Transfer of file data between client and server takes place in units of file blocks. Usually, File is stored in continuous blocks of fixed size on secondary storage. In this model, no need to transfer entire file to client machine and hence, it is better alternative approach to diskless workstations.
- But, if client needs access to entire file then all blocks needs to be transferred and hence, it generates more network traffic and has poor performance. NFS, LOCUS and Sprite use this transfer model.

Byte-level Transfer Model

In this model, transfer of file data between client and server takes place in units of bytes. It offers flexibility as any range of data bytes within file can be requested and hence, it is difficult to manage cached data due to its variable length. Cambridge file server uses this model.

Record-level Transfer Model

This model is applicable to structured files only. Hence, records are transferred between client and server. Research storage system (RSS) uses this model.

6.5 File-Caching Schemes

- These schemes are basically for retaining the cached data of recently accessed files in main memory in order to reduce repeated disk accesses for the same data. It gives good performance due to reductions in disk transfers.
- Hence, it is good approach to improve performance of the file system. It also helps to achieve scalability and reliability as remote data is possible to cache on client node. Therefore, file caching is used by most of DFS.
- File caching scheme for DFS should address the decisions like cache location, propagation of modifications and validation of cache.

6.5.1 Cache Location

- Suppose the original location of the file is server's disk. Following three possible locations can be there to cache the data.

- **Server's main memory :** This is no caching scheme before client can access the file, the file is first transferred from server's disk to the server's main memory and then to the clients main memory across the network. In this, one disk access and one network access is required. It results in good performance as it eliminates disk access. It is easy to implement. As file resides on the same server machine, it is easy to maintain consistency between original file and cached data. The drawback is that, client has to carry out network access for each file access. Hence network cost is involved. It is not scalable and reliable approach.
- **Client disk :** It involves disk access cost at client machine on cache hit. It eliminates network access. In case of crash, data remains in client disk and hence, no need to access again from server for recovery. There is no loss of data as disk is permanent storage. Hence, it offers reliability. Disk also has large storage capacity compared to main memory, resulting in higher hit ratio. Most of the DFS uses file level transfer for which caching in disk is better solution as disk has large storage space for file. The drawback is that, this policy does not work for diskless workstations.
- **Client's main memory :** It works for diskless workstations and avoids network access cost and disk access cost. It contributes scalability and reliability as access request is served locally on cache hit. It is not preferable compared to client disk cache if increased reliability and large cache size is required.
- Cache location can either be main memory or disk. If cache is kept in main memory then modifications done on cached data will lost due crash. If caches are kept in disk then they are reliable. No need to fetch the data during recovery as data resides on disk. Following are the advantages of main memory caches.
  - It permits for diskless workstations
  - Data access takes less time from main memory compared to access from disk.
  - Performance speed up is achieved with larger and inexpensive memory which technology demands today.
  - To speedup I/O server caches are kept in main memory. If both server caches and user caches are kept in main memory then single caching mechanism can be used for both.
- Many remote-access implementations take the hybrid approach considering both, caching and remote service. In NFS, for example, the implementation is based on remote service but is improved with client- and server-side memory caching for performance.

6.5.2 Modification Propagation

- If caching of the data is at client side then file's data may simultaneously be cached on multiple client nodes. If all these cached data copies are exactly same then caches are consistent. If one client changes file data and corresponding data on other client nodes are not changed then caches becomes inconsistent.
- To maintain consistency between all the copies of file data, several approaches are available. These approaches depend on the schemes used for following cached design issues for DFSs.
  - When to propagate modifications made to cached data to file server.
  - How to verify validity of cached data.
- System's performance and reliability depends on the policy used to write back updated data to the master copy of file which is available on server.
- Following policies are used :

Write-through policy

- This policy sends modified cache entry immediately to the server for updating the master copy of the file. This policy is more reliable as little information is lost when client system crashes. The write performance is poor as each write access has to wait until the information is sent to the server.
- The advantages of data caching are only for read accesses as remote service method is used for all write accesses. It is suitable in cases where read to write access ratio is quite large.

Delayed-write policy or write-back caching

- In this policy there is delay in writing the modifications to the master copy on server. Modifications are written first in cache and then later time is done on master copy. First advantage of this policy is that write accesses complete in less time as writes are made to cache.
- Second, the data which may be overwritten prior to writing back on master copy, so last update needs to be written. The limitation of this policy is that, if client crashes then unwritten data are lost and hence less reliable.
- There are variations of delayed write policy. One choice is to flush a block as soon as it is set to be ejected from the client's cache. This alternative can lead to good performance, but some blocks can exist in the client's cache a long time before they are written back to the server.
- A negotiation between this choice and the write-through policy is to scan the cache at regular intervals and to flush blocks that have been modified since the most recent scan. So far one more variation on delayed write is to write data back to the server when the file is closed. This **write-on-close** policy is used in Andrew File System (AFS).

6.5.3 Cache Validation Schemes

- Client machine always use cached data for accesses which is consistent with master copy at server. If client determines that its cached copy is out of date, then it should cache the up-to-date copy of data.
- Following two approaches are used :

Client Initiated Approach

- The client starts a validity check in which it contacts the server. In this check, client checks consistency of its cached data with the data in master copy. The resulting consistency semantics is decided by frequency of validity checking.
- Validity checking can be carried out prior to every access or on first access to a file or at regular intervals. The validity check is carried out by comparing time of last modification of the cached version of data with server's master copy version. If two are same the cached data is considered as up to date. Otherwise recent version is accessed from the server.

Server Initiated approach

- The server maintains the records of the files or portion of files that each client catches. Server must take immediate action whenever it detects a potential inconsistency. If two different clients in conflicting modes cache a file then there is possibility of inconsistency.

- A notification should be sent to server when a file is opened, and the intended read or write mode must be specified for every open. Whenever server detects that file has been opened simultaneously in conflicting modes, it can take action by disabling caching for that particular file. When catching is disabled then a remote-service mode of operation becomes active.
- This policy has some problems also. It violets traditional client server model. Hence, it makes code for client and server programs irregular and complex. It also requires stateful file server which has limitations compared to stateless file servers in case of failures. A validation is carried out with check on open policy which is client initiated approach; it should be used along with server initiated approach.

Comparison of Caching and Remote Service

Sr. No.	Caching	Remote Service
1.	It allows to serve remote accesses locally so that they can be faster as local accesses	Remote access is handled across the network so slower compared to caching.
2.	It reduces the network traffic, server load and improves scalability as server is contacted occasionally.	It increases network traffic, server load and degrades performance.
3.	In case of caching, for transmission of big chunks of data, overall network overload required is at lower side compared to remote service.	In case of remote service, transmission of series of responses to specific requests involve higher network overhead as compared to caching.
4.	Caching is better in case of infrequent writes but if writes are frequent then mechanism used to overcome consistency problem incur large overhead in terms of performance, network traffic, and server load.	In remote service, there is always communication between client and server to have a master copy consistent with client's cached copy.
5.	Caching is better option for machines with disk or large main memory.	If machines are diskless and with small main memory then remote service should be carried out.
6.	In case of caching, the lower-level inter-machine interface is different from the upper-level user interface.	The remote service concept is just an extension of the local file-system interface across the network.

6.6 File Replication

- Replicating the file on many machines improves availability. If nearest replica is used then service time also reduces. The replica of the same file should be kept on failure independent machines so that availability of one replica is not affected by availability of remaining other replica. Replication of files should be hidden from users.
- It is the responsibility of naming scheme to a replicated file name to a particular replica. Higher levels should remain invisible from existence of replicas.

At lower-level different lower-level names are used for different replicas. Replication control such as determination of the degree of replication and placement of replicas should be provided to higher levels. As one replica updates then from user's point of view, the changes should be reflected in other copies as well.

6.6.1 Replication and Caching

- In replication, replica is created at server, whereas cached copy is associated with clients.
- A replica is more persistent as compared to cached copy. Replica is widely known, secure, accurate, available and complete.
- Cached copy existence depends on locality in access patterns. Existence of replica depends on availability and performance.
- Cache copy is dependent on replica. It needs to be periodically verified with replica then it becomes useful.

6.6.2 Advantages of Replication

Following are the advantages of replication :

1. **Increased Availability** : Replication offers availability of the system. Although failure occurs, the system remains operational and available to users. The critical data can be replicated on several servers. If primary copy fails, still it can be accessed from other server.
2. **Increased Reliability** : As several copies of the files are available on different servers, recovery in case of failure is possible. Permanent loss of data due to catastrophic failure is also easy to recover from other replicated copy. Hence, replication offers reliability.
3. **Improved Response Time** : Replication allows data to be accessed locally or from the node whose access time is less than that of primary copy access time.
4. **Reduced Network Traffic** : If replica of file is available with file server that resides on client machine then client access request is serviced locally. Hence, it results in reduced network traffic.
5. **Improved System Throughput** : As different client's requests are serviced by different servers in parallel, it results in improved throughput.
6. **Better Scalability** : If file is replicated at multiple file servers then all the client's requests for that file would not arrive at one file server. In this way, load gets distributed and different client's requests are serviced by different servers. It improves scalability.
7. **Autonomous Operation** : All the files needed by clients for limited time period can be replicated on file server that resides on client machine. This ensures temporary autonomous operation of client node.

6.6.3 Replication Transparency

- User should not be aware about file replication. Although file is replicated, it should appear as a single logical file to its user.
- There should be same client interface for replicated and non-replicated file as well. Following are the two important issues related to replication transparency are naming of replicas and replication control.

Naming of Replicas

- As immutable objects are easily supported by kernel, single identifier can be assigned to all the replicas of immutable object. As all copies are identical and immutable, kernel can use any copy. In case of mutable objects, all copies may not be consistent at particular instance of time.
- If single identifier is assigned to all replicas of mutable object then kernel cannot decide which replica is most up-to-date. Therefore consistency control and management for mutable objects should be carried out outside the kernel.
- Naming system should map a user supplied identifier to the appropriate replica of mutable object. In case if all the replicas are consistent then mapping must provide location of the replicas and their distance from client node.

Replication Control

- Replication control is transparent from user and handled automatically. Replication can be carried out system automatically or can be carried out manually.
- In **explicit replication**, users control the process of replication. Created process specifies server on which file should be placed. If needed then additional copies are created as per request from users. Users also have flexibility to delete one or more replica. In implicit replication, entire process of replication is controlled by system automatically. Users remain unaware of this process. Server to place the file is selected by system automatically. System also creates and deletes replicas as per replication policy.

6.6.4 Multicopy Update Problem

As replicas of file exist on multiple machines, it is necessary to keep all the copies consistent. If update takes place on one copy, it must be propagated to all other copies. Following approaches are used :

Read-Only Replication

In this approach, only immutable files are replicated as they are used only in read only mode. These file gets updated after longer period of time.

Read-Any-Write-All Protocol

This protocol allows replication of mutable files. In this approach, read operation is performed on any copy of the file but write operation is performed by writing to all copies of file. Before performing update to any copy, all copies are locked, then they are updated, and finally locks are released to complete the write.

Available-Copies Protocol

- In read-any-write-all protocol, if server with replicated copy is down at the time of write operation then write cannot be performed. Available-copies protocol allows this operation. In this approach, read operation is performed by reading any available copy of the file but write operation is performed by writing to all available copies of file.
- The assumption in this protocol is that, down server when recovers, it brings its state up-to-date from other server's copies. This protocol provides high availability but does not prevent inconsistencies in failure of communication links.

Primary-Copy Protocol

- In this protocol for each replicated file, one copy is kept as primary and all other copies are considered as secondary copies. The write operation is carried out only on primary but read operation can be carried out on any copy including primary.
- Server with secondary copy either receives notification of updates from server with primary copy or updates are requested by server with secondary copy itself. Primary copy immediately informs updates to secondary copies if any occurs to it.

Quorum-Based protocol

- This protocol is applicable in network partition problem where replicated file are partitioned in two more active groups. All above protocols discussed has some restrictions.
- Following are the definitions of read and write quorum.
  - (i) **Read quorum :** To read a file, if minimum r copies of replicated file F have to be consulted out of n replicated copies of F then set of r copies is called as read quorum.
  - (ii) **Write quorum :** To carry out write operation on the file, if minimum w copies of replicated file F have to be written out of n replicated copies of F then set of w copies is called as read quorum.
- Restriction in this protocol is that sum of r and w should be greater than n (r+w>n). It ensures that, between any pair of read-quorum and write-quorum, there is at least one copy common which is up-to-date. This protocol needs to identify current updated copy to in order to update other copies of the file. This problem is resolved with assigning the version number to copy when it gets updated. Highest version number copy in quorum is current updated copy. New version number to be assigned is one more than the current version number.
- In **read** operation, only highest version number copy is selected to read from read quorum. For **write** operation, only highest version number copy is selected from write quorum to carry out write operation. Before performing write, the version number is incremented by one. Once update is carried out, the new update and new version number is written to all the replicas.
- Consider n = 8, r = 4, and w = 5. In this example, r+w>n condition is satisfied. If write operation is carried out on write quorum {3, 4, 5, 6, 8} then these copies are updated versions with new version number. If read operation is performed on read quorum {1, 2, 7, 3} then copy 3 is common copy as read operation must contain at least one copy of previous write-quorum. This exactly ensures copy 3 as having largest version number from read-quorum. Hence, read is carried out with copy 3.
- Following are few special alternatives suggested for this protocol :
  - **Read-Any-Write-All Protocol :** Suitable to use when read to write ratio is high in case of r = 1 and w = n.
  - **Read-All-Write-Any Protocol :** Suitable to use when write to read ratio is high in case of r = n and w = 1.
  - **Majority-Consensus Protocol :** Suitable to use when read to write operations is nearly 1. In this protocol, read and write quorum is of same size or of nearly equal size. If n=12 then r = 6 and w = 7, or If n=11then r = 6 and w = 7.
  - **Consensus with Weighted Voting :** In this protocol, different replicas are assigned with different votes considering reliability and performance.

- If replica X is accessed more frequently then more votes are assigned to it. Size of quorum depends on replicas selected for quorum. In this protocol, to ensure non-null intersection of read and write quorums, condition $r + w > v$ where v is total number of votes.

6.7 Case Study : Distributed File Systems (DFS)

- DFS forms basis for many distributed application and sharing of data is fundamental to it. DFS supports to share data by multiple processes over long period of time while offering security and reliability.
- SunMicro system's Network File System (NFS) and Andrew File system (AFS) are the examples of distributed file system.

6.7.1 Network File System (NFS)

- NFS is developed by Sun Microsystems and it is used on Linux to join file systems of different computers into one logical whole. Version 3 of NFS was introduced in 1994. NSFv4 was introduced in 2000 and offers a number of improvements over the previous NFS architecture.
- NFS allows a number of clients and servers to share a common file system. These clients and servers can be on the same LAN or in wide area network if server is located geographically at long distance. NFS permits every machine to be both a client and a server simultaneously.

NFS Architecture

- NFS uses remote file service model. Remote server manages all the accesses of clients. NFS offers interface to clients similar to conventional local file system to access the files transparently. All the file operations are implemented by server but their interfaces are offered to the clients. It is **remote service model**. NFS also supports **upload/download model** in which client downloads the file for operations and then uploads it to the server so that updated file can be used by other clients.
- UNIX based version of NFS is predominant version. Client uses system calls of its local operating system (UNIX). The local UNIX file system interface is replaced by interface to virtual file system (VFS). The operations carried out to VFS interface is either passed out to local file system or to **NFS client** component. NFS client then performs remote procedure call (RPC) to access files at remote server.This means NFS client implements file operations as RPC to remote server. VFS hides the differences between different file systems.
- NFS server on server side handles incoming requests. RPC server stub unmarshals these requests and NFS server convert them to regular VFS file operations that are afterward passed to the VFS layer. VFS implements local file system where actual files reside. In this way, NFS is independent of local file systems, provided, if local file system compliant with file system model of NFS.

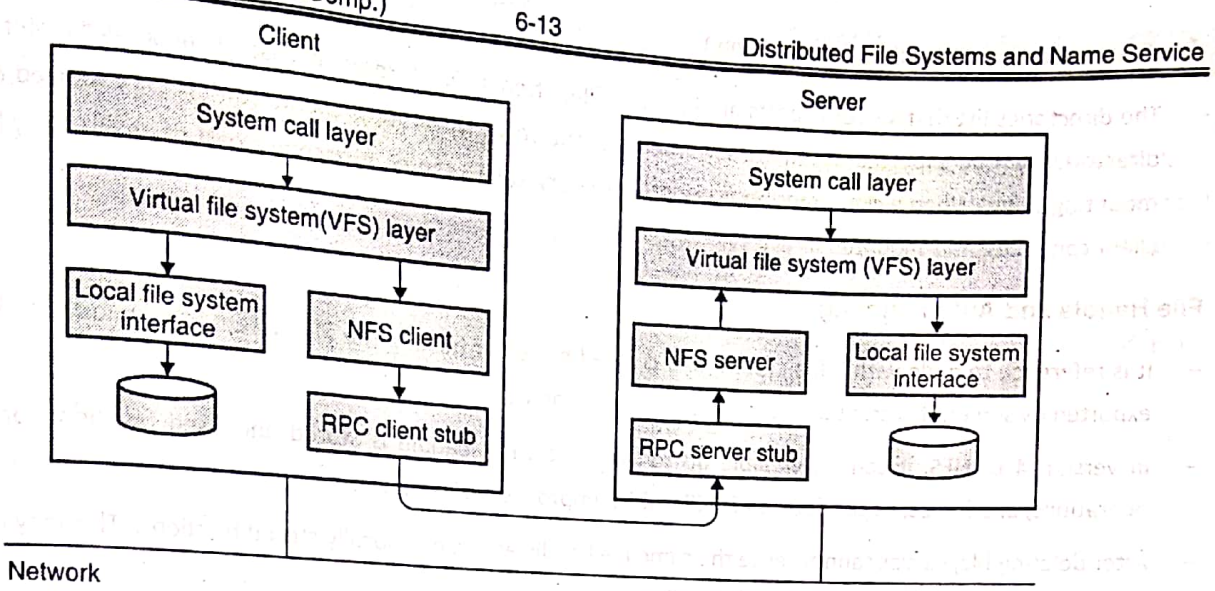


Fig. 6.7.1 : Basic NFS Architecture for UNIX Systems

File System Model, Communication and Naming

- In NFS files are considered as uninterpreted sequence of bytes. Files are organized in naming graph in which nodes represent files and directories. NFS supports hard link or symbolic links just like UNIX file systems.
- Files are accessed by means of UNIX like file handle. At first, client searches file name by using naming service to obtain the file handle. Each file has a several attributes whose values can be found and changed. These attributes includes length and type of file, It's file system identifier, and last time file was modified.
- For reading the data from file, read operation is used. Client specifies offset and number of bytes to read. Writing data to file is carried out with write operation. The position and number of bytes to write is specified by client.
- For communication, NFS protocol is placed on top of RPC layer. This is due to independence of NFS on OS, network architecture and transport protocols. Open network computing remote procedure call **(ONC RPC)** is used for the communication. NFS supports to group several RPC requests in one request in order to reduce number of messages to be exchanged.

This is called as **compound procedures** which do not have any transactional semantics. All the operations in compound procedure is executed in order of their requests. Conflicts cannot be avoided in case if same operations are invoked by other clients simultaneously.

In NFS, stateless server was implemented initially which cannot supports to lock a file for operations. Therefore, a separate lock manager is used to handle same situation. In later version, stateful approach is implemented to support working across wide area network so that clients can use caches effectively.

NFS naming model offers transparent access for remote file system at server to its requesting clients. NFS allows clients to mount part of file system at server. The exported directory by server can be maintained in client's local space. The drawback of this approach in DFS is that it does not allow sharing files.

Remote clients access the NFS server's exported directories. Every NFS server exports one or more of its directories. These directories then accessed by remote clients. Particular directories along with its subdirectories, so in fact entire directory trees are usually exported as a unit.

- The directories list that server exports are kept in a file, often in */etc/exports* in case of Linux. As a result of this, these directories can be exported automatically whenever the server is booted. Clients can access exported directories by mounting them. When a client mounts a remote directory at NFS server, it becomes part of its directory hierarchy and client can access this mounted directory.

File Handle and Automounting

- It is reference to a file within file system and created by server hosting the file system. It is unique to all file systems exported by server. It is created at the time of creation of file.
- In version 4 of NFS, it can be variable up to 128 bytes. File handle is stored and used by client for most of the operations, and hence, avoids look up for file which improves performance.
- After deleting file, server cannot reuse the same file handle as it can be locally stored by client. This may lead to wrong file access by using the same file handle by client.
- Iterative look up with not permitting look up operation to cross a mount point leads to the problem in getting initial file handle. To access the file in remote file system, client must provide file handle of directory where look up would take place. This also requires providing name of the file or directory that is to be resolved.
- NFS version 4 solves this problem by offering a separate operation *pootrootfh* that informs server to solve all file names relative to root file handle of the file system it manages. In NFS, on demand mounting of remote file system is handled by automounter which runs as separate process on client machine.

File Attributes

- In NFS version 4, attributes are classified as mandatory attributes that every implementation must support, set of recommended attributes which should be preferably supported and additional set of named attributes. The named attributes are encoded as an array with entry as attribute-value pair. These are not part of NFS protocol. Following are the examples of some mandatory attributes.
  - **TYPE :** Type of file (regular, directory, symbolic link).
  - **SIZE :** Length of file in bytes
  - **CHANGE :** Indicator for client to check if and/or when the file has changed.
  - **FSID :** Server-unique identifier of file's file system.
- Following are some of the general recommended file attributes :
  - **ACL :** Access control list associated with file.
  - **FILEHANDLE :** Server provide file handle of this file
  - **FILEID:** File system unique identifier for this file.
  - **FS-LOCATIONS :** Locations in network where this file system can be found.
  - **OWNER:** Owner of the file.
  - **TIME_ACCESS :** Last time of accessing the data of file.
  - **TIME_MODIFY :** Time when file data were last modified.
  - **TIME_CREATE :** Time when file was created.

Synchronization

- Distributed file system allows to share its files among multiple clients. These shared files between multiple users should remain consistent. For this purpose, synchronization is needed. In NFS file locking is carried out by separate lock manager. In version 4 of NFS file locking is integrated in NFS file access protocol. It may happen that client or server may fail while locks are still held. Recovery mechanism should handle such problem to ensure consistency.
- Four locking operations are defined in NFS version 4 for locking. Same file can be shared by multiple clients if they only read data. It is necessary to obtain write lock to access to modify part of the file data.
- Following operations are provided for locking :
  - **Lock :** Create lock for range of bytes.
  - **Lockt :** Test whether conflicting lock has been granted.
  - **Locku :** Remove a lock from range of bytes.
  - **Renew :** Renew a lease on specified lock.
- Lock is nonblocking operation and used to request a read or write lock on consecutive range of bytes in file. In case of conflicting lock, lock cannot be grated and client has to poll the server later time. Once conflicting lock has been removed, server grant the next lock to the client at top of requesting FIFO list maintained at server side. Lockt is used to check whether any conflicting lock exist. Removing lock id done by using Locku. if client does not renew lease on acquired lock, server will automatically removes it.

Replication and Consistency

- In NFS version 4, cache consistency is handled in implementation-dependent manner. Client has memory cache to hold data read from server. In extension to memory cache, disk cache also exists in client machine. Client caches file data, attributes, and directories and file handles. Client caches data obtained from server while performing several read operations. Several clients on the same machine may share the cache. If modifications are done on data then cached data is flushed back to server.
- If part of file data is cached, then whenever client opens previously closed file, it should revalidate it. Server may hand over some rights to client so that client can locally handle open, close operations. Sever is in charge of checking whether opening file by client should succeed or not.
- Clients can also cache attribute values which can be different at different clients. Modifications to attribute value should be immediately forwarded to server. Same approach is used for file handles and directories.
- NFS version 4 offers minimum support for file replication. Only replication of whole file system is possible.

Fault Tolerance and Security

- RPC mechanism in NFS does not ensure guarantee regarding reliability. It lacks in detecting the duplicate messages. In case of loss of server replay, server will process retransmitted request by client. This problem is solved by means of duplicate-request cache implemented by server. Each client request carries transaction identifier (XID) that is cached by server when request arrives at server. After processing the request, server also caches reply.

- After timer at client expires before reply comes back then client retransmits same request with same XID. Three cases occur. If server has not yet completed original request, it ignores retransmitted request. In other case, server may get retransmitted request after reply sent to client. If arrival time of retransmitted request and time at which reply sent are nearly equal then server ignores retransmitted request. If reply is really did get lost then cached reply is sent to client as reply to retransmitted request.
- In case of locks, if client is granted a lock and it crashes. In this case, server issues lease on ever lock. If not renewed by client then server removes lock freeing the resources held by lock. In case of server failure, a grace period is provided to server after it recovers. In this grace period, clients can reclaim same lock that was previously granted to them.
- As a security measure, older NFS used Diffi-Hellman key exchange to establish a session. NFS version 4 uses authentication protocol Kerberos. RPCSEC_GSS secured framework is also supported foe setting up secured channels. Authorization in NFS is analogues to secure RPC. ACL file attribute is used to support access control.

6.7.2 Andrew File System (AFS)

- UNIX programs can transparently access the remote shared files. Like NFS this transparency is offered by Andrew file system (AFS) to UNIX programs. Normal UNIX primitives are used by UNIX programs to access AFS files without carrying out any modifications or recompilation. AFS is compatible with NFS. File system in server is NFS based. Hence, File handles are use for reference of file. Remote access to file is provided via NFS.
- Scalability is major design goal of AFS. It supports to large number of active users. The whole file is cached at client node. Following are two design characteristics.
  - **Whole-file serving :** The entire content of directory and file are transferred to client machine by AFS servers.
  - **Whole-file caching :** This transferred file by server is then stored on the local disk which is permanent storage. Cache contains several recently used files. Open request are carried out locally.
- Following is the operation of AFS :
  - If file is not available on client machine. Suppose user process issues open system call for the file in shared file space. Now server is located and request is sent for copy of file.
  - The copy is then stored in local UNIX file system in client machine. The copy is then opened and its file descriptor is sent to client.
  - Processes in clients perform operation on local copy of the file. When process in client issue close system call, if the local copy is updated then its content is sent to server. Server updates the file content and timestamp on the file. The local copy is stored on client's disk for future use.

Implementation

- AFS implementation contains two software components which are UNIX processes called as Vice and Venus. Vice process runs on server machine as user-level UNIX process.
- Venus process runs on client machine as user-level UNIX process. At client side, local and shared files are available. Local files are handled as normal UNIX files. Shared files are stored on server and cached by clients in disk cache.

- UNIX kernel on each client and server is modified version of BSD UNIX. These modifications are done to interpret open, close and other system calls when they refer to files in shared name space and pass them to Venus process in client's machine. At client side, one of the file partitions is used as cache storing files cached copies of files from shared space. The management of cache is carried out by Venus process.
- It removes LRU (least recently used) files so that new files accessed from server gets space. The size of disk cache at client is large. Vice server implements the flat file service. Venus processes at client side implements hierarchic directory structure that is needed by UNIX user programs. Each file in shared file space is assigned with 96-bit file identifier. These files are identified by this identifier. It is the job of Venus process to translate path name to file identifier.

Cache Consistency in AFS

- Vice process at server side provides call back promise whenever it transfers file to Venus process running at client side. In fact, call back promise is token that guarantees that, whenever any client updates file copy, it will be notified to Venus process.
- These tokens are stored in two states along with cached file on disk cache at client machine. These are : valid and cancelled state. When server carries out a request to modify file, it notify to each client (Venus processes) to which token was issued. This notification (call back) is RPC from server to Venus process. After receiving call back by Venus process, it sets the call back promise as cancelled.
- For open operation on file, although file is available in cache, Venus checks its state. If it cancelled state then recent copy of file is fetched from server. For valid state, cached copy is then opened for further operations.
- Vice is user level process and server is dedicated to offer AFS service. UNIX kernel is modified in AFS host so that it can handle file operations by using file handles instead of conventional file UNIX file descriptors.

6.8 Introduction to Name Services and Domain Name System

- Name service is used by client processes to locate objects or to obtain addresses of resources by submitting their names. Name services often used to hold addresses and other information regarding users, machines, network domains and remote objects etc.
- Names are used to refer to resources in distributed system. These resources comprise computers, files, web pages, services and many more. URL is the example of name which is used to access web page. These names of entities are organized in name space.

6.8.1 Names, Identifiers and Addresses

- In distributed system names are used to refer to entity. Name is string of bits or characters. There are many entities exist in distributed system such as messages, web pages, mailboxes, network connections etc. It also includes resources such as files, printers, disks, computers etc.
- In order to operate on these entities, they need to be accessed. To access the entity we need access point. The name of access point is address. When mobile computer changes location, a different IP address is assigned to it. It indicates that entity may change its access time in course of time.

- Address is also a name. Name should be location independent. Name of the entity should be independent from its address.
- Identifier is also a name which is used to identify the entity. Identifier should refer to one entity and each entity should refer to by at the most on identifier. Identifier should always refer to the same entity.
- In many computer systems, addresses and identifiers are represented in the form of bit strings. Human friendly name such as URL is represented as string of characters. Many of the names are specific to some service.

6.8.2 Name Services and the Domain Name System

- Name service maintains collection of one or many naming contexts. Name service supports name resolution. Name resolution is to look up attributes from a given name. As distributed system is open, name management is separated from other services.
- Same naming scheme should be used for the resources managed by different services. Sometimes resources created in different administrative domains can be shared. Therefore common name service is required.
- Originally name services were quite simple as they were designed for single administrative domain. Considering the large distributed system with interconnection of networks, a larger name-mapping is required. Global name service is the example of naming service.
- Global name service and handle service are the two examples of name service that concentrated on scalability to large number of objects. Internet Domain Name Service (DNS) is widely used.

Name Spaces

- Names in distributed system are organized in name space. Name space can be represented as labeled diagraph having two types of nodes. Leaf node represents named entity and it has no outgoing edges. This leaf node stores information about entity such as address. It also stores state of the entity, for example, in file system it contains complete file it is representing. Directory node in name space has number of outgoing edges. Directory node maintains directory table in which outgoing edge represented as pair (edge label, node identifier).
- Naming graph contains root node. The path in naming graph contains sequence of labels. For example, path N:<label 1, label 2, ..., label n> contains N as first node in graph. Such sequence is called as path name. If first name in path name is the root of naming graph then it is called as absolute path. Otherwise it is called as relative path.
- DNS names are called as domain names. They are strings just like absolute UNIX file names. DNS name space has hierarchic structure. A domain name comprises of one or more strings called as labels. They are separated by delimiter "." (dot). There is no delimiter at beginning and end of domain name. Prefix of name is initial section of name. For example *dcs* and *dcs.qmw* are both prefixes of *dcs.qmw.ac.uk*. DNS servers do not recognize relative name. All the names referred to the global root.
- In general, alias is similar to UNIX like symbolic link which allows substituting the convenient name in place of complicated one. DNS permit aliases in which one domain name is defined to stand for another. Aliases provide the transparency. Aliases are generally used to specify machines name on which FTP server or web server runs. Alias is updated in DNS database suppose web server is moved to another machine.

- Naming domain is name space for with there is single administrative authority for assigning names within it. Domains in DNS are collection of domain names. Only some domain names identify the domain. A machine can have same name as domain. Naming data belonging to different naming domains are stored by distinct name servers managed by corresponding authorities

Combining the Name Spaces

- It is possible to embed part of one name space in other name space. Mounting file in UNIX and NFS is the example of the same. The entire UNIX file systems of two different machines can be merged by replacing each file system's root by super root. Then mount each machine's file system in this super root. In this way name spaces can be merged by creating higher level root. It raises the problem of backward compatibility.
- DCE (Distributed Computing Environment) name space permits heterogeneous name spaces to be embedded in it. DCE name space contains junctions similar to mount points in UNIX and NFS. These junctions allow the mounting of heterogeneous name spaces.
- File system mounting allow users to import files from remote server and share them. Spring naming service supports to create name spaces dynamically. It also supports to share individual naming context selectively.

6.8.3 Name Resolution

- Name resolution is iterative process. Example of iterative nature of resolution is use of aliases. If DNS server is requested to resolve an alias www.dcs.qmw.ac.uk, it first resolves the alias to another domain name *copper.dcs.qmw.ac.uk*, which must be further resolved to generate IP address. Hence, use of aliases presents cycles in name space in which case resolution never terminate. As a solution, threshold should be maintained and if resolution process crosses it then terminate the process. In other case administrator may carry out certain actions if aliases creates cycles.
- As DNS database is huge and used by large population. It is not stored on single server. Otherwise server would become bottleneck. Name services that are heavily used should use replication to ensure high availability. Administrative authority of the domain manages data belonging to domain which is stored on local name server. A name server may store data belonging to more than one domain. Local name server also needs to take help of other name servers to resolve the names as all the enquiries cannot be answered by it.
- Process of locating naming data from among several name servers so as to resolve the name is called navigation. The client name resolution software carries out navigation. DNS carries out iterative navigation as shown in Fig. 6.8.1.

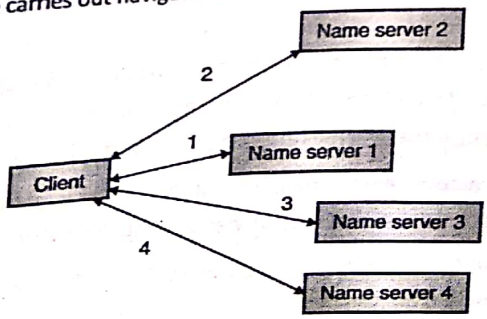
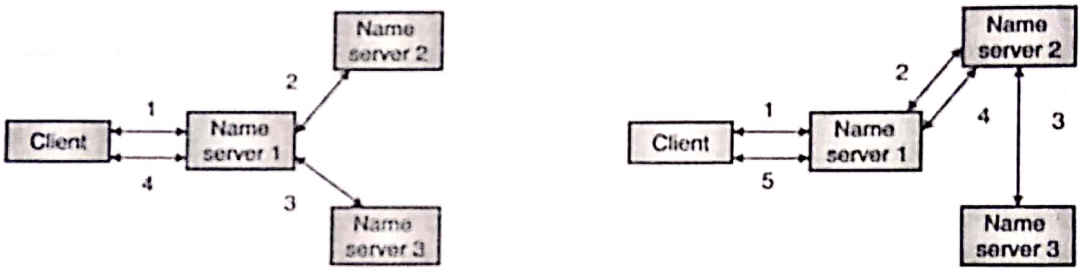


Fig. 6.8.1 : Client iteratively contact name server 1 to 4 in order to resolve name.

- Initially client presents name to local name server. If it has the name, it returns it immediately. Otherwise it will suggest another server which can help. Now resolution proceeds at new server. If needed, further navigation is carried out until the name is located or discovered to be unbound. In multicast navigation client multicast name to be resolved and needed object type to group of servers. Server storing this named attribute only replies to the request.
- In recursive name resolution, name server coordinates the resolution of name and returns result back to user agent. It is further classified as non-recursive and recursive server controlled navigation. In non-recursive server controlled navigation, client can choose any name server.
- This server then either multicast request to its peers or it communicates iteratively with peers. In recursive server controlled navigation, client once more contact single name server.
- If this server does not hold name then it contact to its peer that holds larger prefix of the name, which in turn attempt to resolve it. It is repeated until name is resolved. Fig. 6.8.2 shows non-recursive and recursive server controlled navigation.



(a) Non-recursive server controlled Navigation (b) Recursive server controlled Navigation

Fig. 6.8.2

- The access to name server running in one administrative domain by client running in another administrative domain should be prohibited. In DNS and other name services, client name resolution software and server maintains cache of results of previous name resolution. On a clients request to resolve name, first client name resolution software enquire cache. If recent result of previous name resolution is found then return it to client. Otherwise request is forwarded to server. That server in turn may return result cached from other server.
- Caching improves performance and availability. It saves communication cost and improves response time. High level name servers such as root servers are eliminated due to caching.

6.8.4 Domain Name System

- DNS naming database is used across the internet. The objects named by DNS are computers. For these computers IP addresses are stored as attributes. In DNS domain names simply are called as domains. Internet DNS has bound millions of names. The look ups against these names are carried out from all around the world.
- Mainly DNS is used for naming across the Internet. DNS name space is partitioned organizationally and as per geography as well. In name highest level domain is mentioned at right. Following are the generic domains.
  - ***com :*** Commercial organizations
  - ***edu :*** Educational institutions and universities
  - ***gov :*** US Governmental agencies

- ***mil :*** US military Organizations
- ***net :*** major network support centers
- ***org :*** Organizations not listed in above domains
- ***int :*** International organizations

- Every country has its own domain. Some of the examples are :
  - ***us:*** United states
  - ***fr:*** France
  - ***in:*** India
- Countries other than USA use their own domain. For example, India has domain *ac.in* which corresponds to *edu* domain.

DNS Queries

Internet DNS is used for simple host name resolution and looking up electronic mail host :

- **Host name resolution :** In this, applications use DNS to resolve host names in to IP addresses. URL contains domain name. When it is given to browser, it makes DNS enquiry and obtains IP address. Browsers use http to communicate with web servers at an IP address with a reserved port number.
- **Locating the mail host :** Electronic mail software uses DNS to resolve domain names in to IP addresses of mail host.

DNS Name Servers, Navigation and Query Processing

- Scaling is achieved with combination of partitioning the naming database and by replicating and caching the part of this database close to the point of requirement. DNS database is distributed across logical network of servers. Each server holds part of database mainly of local domain so that local request can be fulfilled locally within the domain.
- Each server also records domain names and addresses of other servers. DNS naming data is divided into zones where each zone contains following data.
  - It contains attribute data for names in domains and less the data about sub-domains. For example, it could contain data for organization (*college.ac.in*) and less the data about department (*department.college.ac.in*).
  - Names and addresses of at least two servers in zone which maintains trustworthy and reliable data for the zone.
  - Names of name servers that holds data for delegated sub-domains and data which gives IP addresses of these servers quickly.
  - Parameters related to zone management that governs the replication and caching.
- Any server can cache data from other servers so that it can be required in future name resolution process. Each entry in zone contains time to live value so that cached data from un-authoritative server by client will remain useful. In this case, this un-authoritative server caches data from authoritative server. If client sends query after time to live period then un-authoritative server contact again to authoritative server. This minimizes network traffic and offers flexibility to system administrators.
- DNS can be used to store arbitrary attributes. Type of query specifies what is required. It can be IP address, name server, mail host or other information.

- A DNS client is resolver which is implemented in library software. It is simplest request–reply protocol. Both iterative and recursive resolution is supported by DNS and client side software specifies the type of resolution to be carried out.
- Name servers store the zone data in files in the form of resource records. Following are some examples of resource records for Internet database.

Type of Record	Associated Entity	Contents	Meaning
A	Host	IP address	IP address of this computer
MX	Domain	List of <preference, host pair>	Refers to mail server to handle mail addressed to this node
PTR	Host	Domain name	Holds canonical name of host
TXT	Any kind	Arbitrary text	Text string
SOA	Zone	Parameters governing zone	Holds information on represented zone
NS	Zone	Machine architecture and operating system	Holds information on the host this node represents.
CNAME	node	Domain name for alias	Symbolic link.

- Berkeley Internet Name Domain (BIND) is an implementation of DNS for machines having UNIS OS running on them. Client programs link in library software as the resolver. DNS name servers run named daemon. BIND permits three servers which are primary, secondary and caching-only servers. The named program implements just one of these as per configuration file contents.
- Typically, organization has one primary, one or more secondary that offer service for name serving on different LANs at the site. In addition to this, caching-only servers are run by individual machines to minimize network traffic and speed up the response time.

6.9 Directory Services

- Directory services are attribute based naming systems. Directory service stores binding between names and attributes and that look up entries matches with attribute based specifications called as directory service. Some of the examples are LDAP, X.500 and Microsoft's Active Directory Services. Directory service returns attributes of any objects. Attributes are more powerful compared to names. For example, Programs usually can be written to select objects by their attributes and not names.
- A discovery service is a directory service. This discovery service registers services provided in spontaneous networking. In spontaneous networking devices gets connected at any moment of time without any warning. There is no any administrative preparation carried out for these devices when they connect in network. It is required to support set of clients and services to be registered transparently. There should not be any human intervention for the same.
- To support this, discovery service offers interface for registering and de-registering these services automatically. It also offers interface for the clients to look up these services. For example, customer in hotel should be able to connect his laptop to printer automatically without configuring it manually.

- This is possible if laptop uses look up interface of discovery service that finds available network printers that fulfills users need. In this case required attributes of the printing service may specify whether "laser" or "inkjet", offers color printing or not, location of printer etc.
- Jini system is developed for spontaneous networking. It assumes JVM runs in all the machines and machines communicate with each other through remote method invocation. It offers facility to discover the service, for transaction, for shared da places, and for events. Discovery related components in Jini system are look up services, Jini clients, and Jini services. Jini client uses look up service. To locate the look up service, client multicast to well known IP multicast address. Look up services listen on a socket bound to the same address in order to receive such requests.

6.10 The Global Name Service (GNS)

- Global Name Service (GNS) was designed at DEC Systems Research Center. It offers facility for resource location, authentication and mail addressing. Following are the design goals of GNS.
  - To handle arbitrary number of names and to serve arbitrary number of organizations.
  - A long life time.
  - High Availability.
  - Fault isolation: Local failure does not affect the entire system.
  - Tolerance of mistrust.
- These goals indicate that any number of computers, users can be added in system and hence, it considered the support for scalability. If organizational structure changes then the structure of name space may also change. The service should also accommodate the changes in the names of individuals, organization etc.
- Considering the large size of distributed system and naming database, caching is used in GNS. It assumes that, changes in database will occur infrequently and hence slow propagation of updates is adopted. Client can detect and recover from use of out of date naming data.
- GNS naming database is composed of tree of directories which holds names and value. Path names are used to refer the directory similar to UNIX file system. Each directory is assigned with unique directory identifier (DI). A directory holds list of names and references. The leaves of directory tress also hold values which are further organized in value tree.
- In GNS, names consist of two parts : <directory name, value name>. First part identifies directory and second refers to the value tree. The attributes of user Rajesh in directory C would be stored in value tree named as <DI of Root directory/A/B/C, Rajesh>. The password in value tree can be referenced as :

  <DI of Root directory/A/B/C, Rajesh/password>
- The directory tree is partitioned and stored in many servers. Each partition is again replicated to many servers. If two or more concurrent updates take place then consistency of tree is maintained. Two directory trees can be merged by inserting new root above roots of these two trees to be merged. This problem is solved by unique directory identifier. Whenever client uses the name </A/B/C, Rajesh>, local user agent add prefix to the directory as <#567/A/B/C, Rajesh>.
- User agent then sends this derived name to GNS server. In the same way, user agent deals with relative name referring to working directories.

- GNS maintains table of well-known directories in which it list all the directories which are used as working roots. This table is held in current real root directory of the naming database. Whenever real root of naming data base changes due to adding the new root, all GNS servers are informed about location of new root. In GNS restructuring of database can be done if any organizational changes occurs.

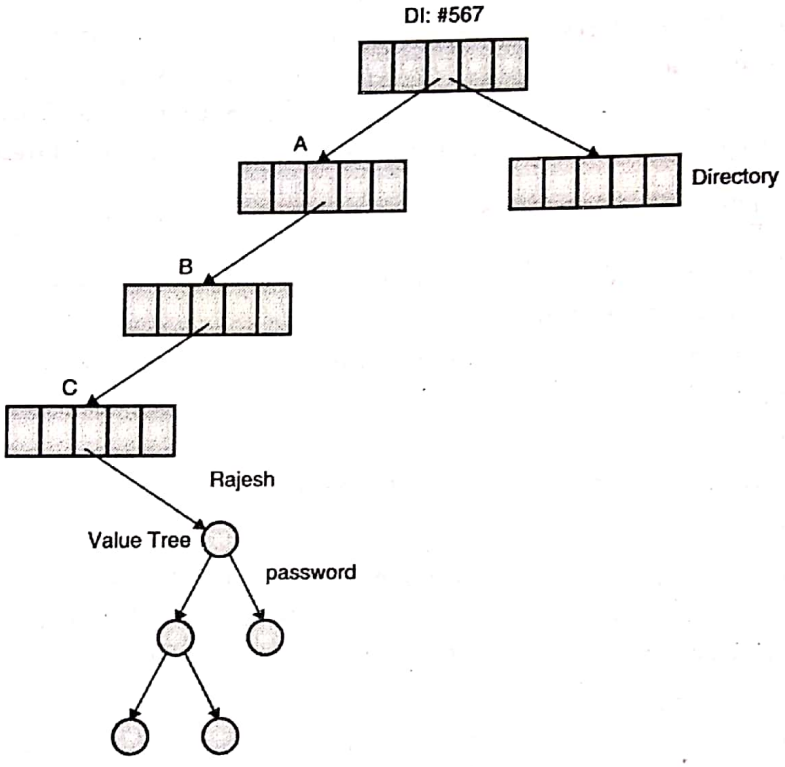


Fig. 6.10.1 : GNS directory tree

6.11 The X.500 Directory Service

- X.500 Directory Service is used to satisfy descriptive queries to look up names and attributes of other users or system resources. The use such service is quite diverse. For example, the enquiries can be for accessing white pages to obtain user's email address or yellow pages query may be for obtaining names and telephone numbers of garages.
- Individuals or organizations can use directory service to provide information about them and resources they wish to offer for use in network. It is possible for users to search directory for information with partial knowledge of its name, structure and contents. ITU and ISO standard organization have defined X.500 Directory Service as a network service.
- It is used for access to hardware and software services and devices. The X.500 servers maintain the data in tree structure with named node just like other name servers. Each node of tree in X.500, stores wide range of attributes. The entries can be searched by any combination of attributes.
- The X.500 name tree is called as **Directory Information Tree (DIT)**. The entire directory structure along with data associated with nodes is called as **Directory Information Base (DIB)**.

- It is intended to have single integrated DIB with information provided by organizations throughout the world. The portion of DIB is located in individual X.500 servers. Clients establish connection with server to access the directory by issuing the requests.
- Client can contact any server. If data are not in the part of DIB of contacted server then it will either invoke other servers or redirects the client to another server. In X.500, servers are called as Directory Service Agents (DSAs) and clients are called as Directory User Agents (DUAs). Following Fig. 6.11.1 shows the service architecture of X.500. It is one of the navigation model in which each DUA client interact with single DSU server which further accesses the other DSU servers to satisfy the client requests.

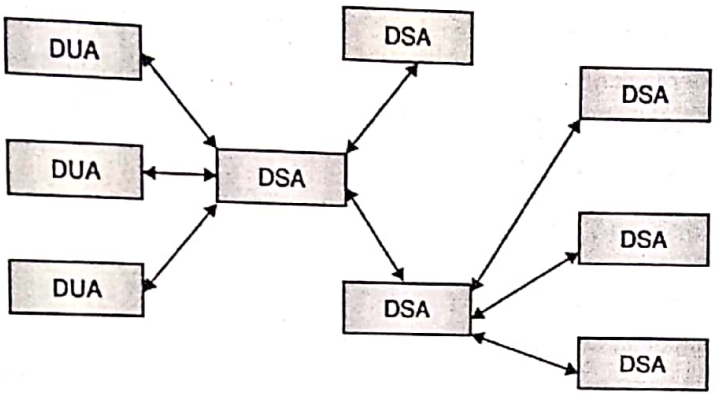


Fig. 6.11.1 : X.500 Service Architecture

- Each entry in DIB consists of name and attribute. The name of an entry corresponds to path from root to entry in DIT. Absolute or relative path names can be used. Also DSU can establish a context that includes base node and then use shorter relative names that gives path from base node to relative entry.
- Directory can be accessed by using two access requests : read and search.
  - ***read :*** An absolute or relative name to an entry is given along with a list of attributes to be read. DSA then navigates DIT to locate the entry. If the part of tree that includes entry is not available on this DSA server, then it passes request to other DSA servers. The retrieved attributes then it returns to client.
  - ***Search :*** This access request is attribute-based, base name and filter expression are supplied as arguments. The base name indicates the node in DIT from which search is to begin and filter expression is to be evaluated for every node below the base node. The search criterion is specified by filter. The search command then returns names for all entries for which filter evaluates TRUE value.
- DSA interface offers operations to add, delete and modify the entries. Access control is also provided for both query and update operation. As a result, access to some part of DIT can be restricted to user or group of users.

6.12 Designing Distributed Systems : Google Case Study

6.12.1 Google Search Engine

- From a distributed systems viewpoint, Google offers attractive extremely challenging requirements, mainly in terms of scalability, reliability, performance and openness. Like any web search engine, Google search engine return an ordered list of the most relevant results that match to the given query by searching the content of the Web. The search engine contains a set of services for **crawling** the Web and **indexing** and **ranking** the searched pages.

- The **crawler** locates and retrieves the contents of the Web and passes the contents onto the indexing subsystem. This is carried out by a software service called ***Googlebot***, which recursively reads a given web page, harvesting all the links from that web page and then scheduling further crawling operations for the harvested links.
- The **indexing** produces an index for the contents of the Web which is on a much larger scale. More specifically, indexing produces an ***inverted index*** mapping words in web pages and other textual web resources onto the positions where they occur in documents, including the accurate position in the document and other related information for example, the font size and capitalization. The index is also sorted to support efficient queries for words against locations.
- Indexing does not provide information about the relative importance of the web pages that contains a particular set of keywords. In **ranking**, higher rank denotes importance of a page and it is used to make sure that important pages are returned nearer to the top of the list of results than lower-ranked pages. Ranking in Google also considers factors like nearness of keywords on a page and their font size (large or small).

6.12.2 Google Applications and Services

- Apart from the search engine, Google now offers a wide range of web-based applications that also includes Google Apps. Now days, cloud computing also extensively supported by Google. It provides software as a service (SAS), which offers application-level software over the Internet, as web applications. A Google Apps is the main example of it. Google Apps includes set of web based applications such as Gmail, Google Docs, Google Sites, Google Talk and Google Calendar.
- Apart from SAS, with new addition of Google App Engine, Google now offers its distributed systems infrastructure as a cloud service. This cloud infrastructure supports all its applications and services, including its web search engine. The Google App Engine now offers external access to a part of this infrastructure, permitting other organizations to run their own web applications on the Google platform. Following are the examples of Google applications :
  - **Gmail :** It is a mail system of Google that hosts messages.
  - **Google Docs :** It is Web-based office suite which supports for editing of documents held on Google servers in shared mode.
  - **Google Sites :** These are Wiki-like web sites having shared editing facilities.
  - **Google Talk :** It offers instant text messaging and Voice over IP.
  - **Google Calendar :** It is Web-based calendar having all data hosted on Google servers.
  - **Google Wave :** It is a collaboration tool integrating email, instant messaging, wikis and social networks.
  - **Google News :** It is automated news aggregator site.
  - **Google Maps :** This App offers scalable web-based world map including high-resolution imagery and unlimited user generated overlays.
  - **Google Earth :** This App offers scalable near-3D view of the globe with unlimited user-generated overlays.
  - **Google App Engine :** It offers distributed infrastructure of Google to other parties outside of Google.
- Google infrastructure provides scalability and it uses approaches to scale in large scale. Google considers scalability problem related to dealing with more data, more queries, and expecting better results.

- Google provides high reliability for the search functionality and Apps it offers to customer. Its service level agreement contains 99.9% guarantees to offer services that includes all the Google applications.
- Google aims for low latency to user interactions to gain high **performance**. Google keeps the target of completing web search operations in 0.2 seconds and achieving the throughput to react to all incoming requests while handling very large datasets. Google also highly supports for **openness** in order to accommodate the wide range of web applications to be developed in future.
- To satisfy above stated requirements, Google has developed the overall system architecture as shown in 6.12.1. As shown in Fig. 6.12.1, the underlying computing platform is at the bottom and the well-known Google services and applications are at the top. The distributed infrastructures offering middleware support for search and cloud computing is provided by middle layer. This infrastructure offers the common distributed system services for developers of Google services and Apps and encapsulates key techniques that handle scalability, reliability and performance.
- This infrastructure bootstraps the development of new applications and services through reuse of the underlying system services and, more subtly, offers largely coherence to the growing Google code base by implementing common strategies and design principles.

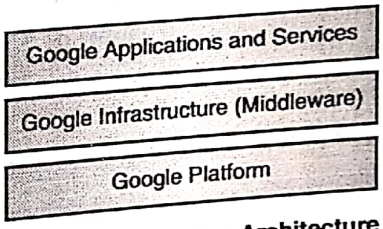


Fig. 6.12.1 : The Overall System Architecture of Google

6.12.3 Google Infrastructure

- Following Fig. 6.12.2 shows Google infrastructure. The system is constructed as a set of distributed services offering core functionality to developers. The *protocol buffers* component offers a common serialization format for Google, together with the serialization of requests and replies in remote invocation. Google *publish-subscribe* service supports the efficient dissemination of events to potentially large numbers of subscribers.

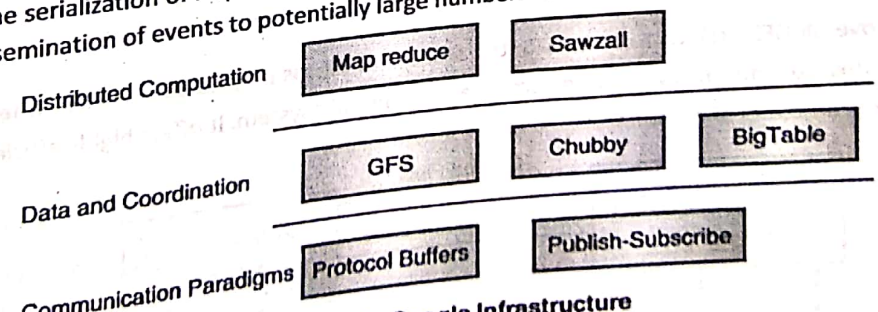


Fig. 6.12.2 : Google Infrastructure

- Data and coordination services offers unstructured and semi-structured\ abstractions for the storage of data coupled with services to support coordinated access to the data : GFS offers a distributed file system optimized for the particular requirements of Google applications and services. Chubby supports coordination services and the ability to store small volumes of data. Bigtable provides a distributed database offering access to semi-structured data.
- Distributed computation services offers means for carrying out parallel and distributed computation over the physical infrastructure : MapReduce supports distributed computation over potentially very large datasets.

- Sawzall provides a higher-level language for the execution of such distributed computations. Communication is supported through RMI, RPC and publish-subscribe service.

6.12.4 The Google File System (GFS)

- The GFS mainly aims at demanding and rapidly growing needs of Google's search engine and the Google web applications. Following are the requirements for GFS :
  - GFS must run reliably on the physical architecture.
  - GFS is optimized for the patterns of usage within Google, both in terms of the types of files stored and the patterns of access to those files.
  - GFS must fulfill all the requirements for the Google infrastructure as a whole.
- GFS provides a conventional file system interface offering a hierarchical namespace with individual files identified by pathnames. Following file operations are supported. The main GFS operations are very similar to those for the flat file service.
  - ***create*** – create a new instance of a file;
  - ***delete*** – delete an instance of a file;
  - ***open*** – open a named file and return a handle;
  - ***close*** – close a given file specified by a handle;
  - ***read*** – read data from a specified file;
  - ***write*** – write data to a specified file.
- The parameter for GFS *read* and *writes* operations specify a starting offset within the file. The API offers, *snapshot* and *record append* operations. The snapshot operation offers an efficient mechanism to make a copy of a particular file or directory tree structure. The *record append* operation supports the common access pattern whereby multiple clients carry out concurrent appends to a given file.

GFS Architecture

- Fig. 6.12.3 shows overall GFS architecture. In GFS, the storage of files is in fixed-size *chunks*, where each chunk is 64 megabytes in size.This size chosen is very large compared to other file system. It offers highly efficient sequential reads and appends of large amounts of data.

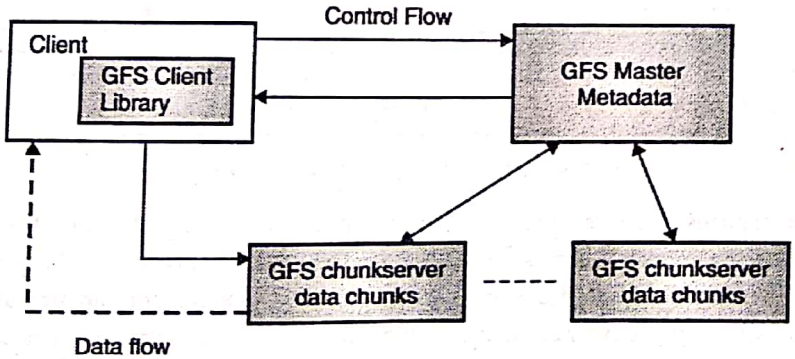


Fig. 6.12.3 : Overall GFS Architecture

- GFS provides a mapping from files to chunks and also supports standard operations on files, mapping down to operations on individual chunks. Figure shows an instance of a GFS file system as it maps onto a given physical cluster. Each GFS cluster has a single *master* and multiple *chunkservers* which jointly offer a file service to large numbers of clients concurrently accessing the data.
- The master manages metadata about the file system. It defines namespace for files, access control information and the mapping of each particular file to the associated set of chunks. All chunks are replicated and location of the replicas is maintained in the master. Replication provides reliability in case if any software or hardware failure occurs. The key metadata is maintained persistently in an operation log. This helps in recovery in the event of crashes.
- The master is centralized which can be single point of failure. The operations log is replicated on several remote machines, so the master can be restored on failure. GFS avoids heavy use of caching and clients do no cache file data. The location information of chunks is cached at clients when first accessed, in order to reduce interactions with the master. GFS does not have any strategy for server-side caching. It uses buffer cache in Linux to maintain frequently accessed data in memory.

Review Questions

Q. 1 Explain in short services provided by distributed file system.

Q. 2 Explain desirable features of good distributed file system.

Q. 3 Explain different file models.

Q. 4 Explain different file accessing models..

Q. 5 Explain different techniques to transfer data between client and server.

Q. 6 Explain different policies to propagate modifications.

Q. 7 Explain different cache validation schemes.

Q. 8 Differentiate between caching and remote service.

Q. 9 Differentiate between caching and replication.

Q. 10 What are the advantages of replication? Explain.

Q. 11 Explain in detail replication transparency.

Q. 12 Explain different approaches to update the multiple copies of file.

Q. 13 Explain quorum based protocol for updating of multiple copies.

Q. 14 Explain in detail network file system (NFS).

Q. 15 Explain architecture of NFS.

Q. 16 Explain in detail Andrew File System (AFS).

Q. 17 What are names, identifiers and addresses? Explain.

Q. 18 Write short note on "Name Spaces".

Q. 19 Explain different name resolution techniques.

Q. 20 Write note on domain name system.

Q. 21 Write note on directory services.

Q. 22 Write note on X.500 Directory Service.

Q. 23 Write note on global name service.

Q. 24 Explain Google applications and services.

Q. 25 Explain the overall system architecture of Google.

Q. 26 Explain Google file system.

□□□