Distributed File Systems and Name Services

- ## Introduction and features of DFS

A Distributed File System (DFS) as the name suggests, is a file system that is distributed on multiple file servers or multiple locations. It allows programs to access or store isolated files as they do with the local ones, allowing programmers to access files from any network or computer.

The main purpose of the Distributed File System (DFS) is to allow users of physically distributed systems to share their data and resources by using a Common File System. A collection of workstations and mainframes connected by a Local Area Network (LAN) is a configuration on a Distributed File System. A DFS is executed as a part of the operating system. In DFS, a namespace is created and this process is transparent for the clients.

DFS has two components:

- Location Transparency

Location Transparency is achieved through the namespace component.

- Redundancy –

Redundancy is done through a file replication component.

In the case of failure and heavy load, these components together improve data availability by allowing the sharing of data in different locations to be logically grouped under one folder, which is known as the "DFS root".

It is not necessary to use both the two components of DFS together, it is possible to use the namespace component without using the file replication component and it is perfectly possible to use the file replication component without using the namespace component between servers.

Features of DFS :

- Transparency :
  - Structure transparency –

There is no need for the client to know about the number or locations of file servers and the storage devices. Multiple file servers should be provided for performance, adaptability, and dependability.

- Access transparency –

  Both local and remote files should be accessible in the same manner. The file system should be automatically located on the accessed file and send it to the client's side.

- Naming transparency –

  There should not be any hint in the name of the file to the location of the file. Once a name is given to the file, it should not be changed during transferring from one node to another.

- Replication transparency –

  If a file is copied on multiple nodes, both the copies of the file and their locations should be hidden from one node to another.

- User mobility :

  It will automatically bring the user's home directory to the node where the user logs in.

- Performance :

  Performance is based on the average amount of time needed to convince the client requests. This time covers the CPU time + time taken to access secondary storage + network access time. It is advisable that the performance of the Distributed File System be similar to that of a centralized file system.

- Simplicity and ease of use :

  The user interface of a file system should be simple and the number of commands in the file should be small.

- High availability :

A Distributed File System should be able to continue in case of any partial failures like a link failure, a node failure, or a storage drive crash.

A high authentic and adaptable distributed file system should have different and independent file servers for controlling different and independent storage devices.

- Scalability :

  Since growing the network by adding new machines or joining two networks together is routine, the distributed system will inevitably grow over time. As a result, a good distributed file system should be built to scale quickly as the number of nodes and users in the system grows. Service should not be substantially disrupted as the number of nodes and users grows.

- High reliability :

  The likelihood of data loss should be minimized as much as feasible in a suitable distributed file system. That is, because of the system's unreliability, users should not feel forced to make backup copies of their files. Rather, a file system should create backup copies of key files that can be used if the originals are lost. Many file systems employ stable storage as a high-reliability strategy.

- Data integrity :

  Multiple users frequently share a file system. The integrity of data saved in a shared file must be guaranteed by the file system. That is, concurrent access requests from many users who are competing for access to the same file must be correctly synchronized using a concurrency control method. Atomic transactions are a high-level concurrency management mechanism for data integrity that is frequently offered to users by a file system.

- Security :

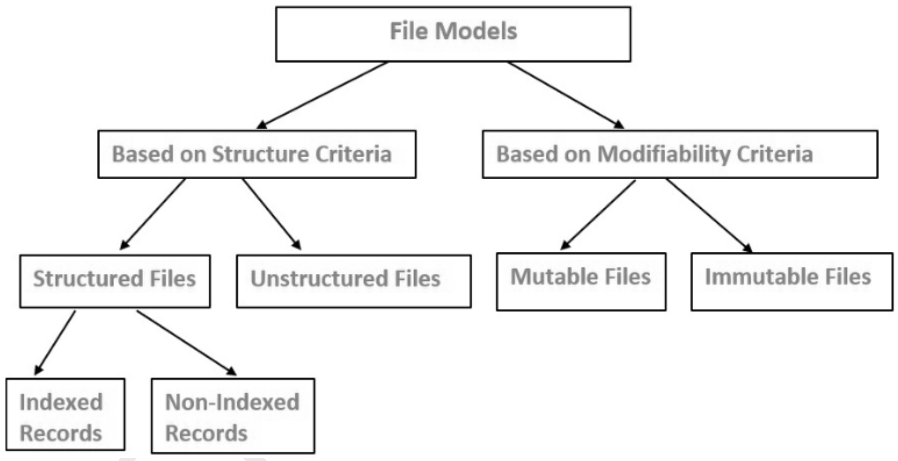
  A distributed file system should be secure so that its users may trust that their data will be kept private. To safeguard the information contained in the file system from unwanted & unauthorized access, security mechanisms must be implemented.

- Heterogeneity :

Heterogeneity in distributed systems is unavoidable as a result of huge scale. Users of heterogeneous distributed systems have the option of using multiple computer platforms for different purposes.

- **File models**

In Distributed File Systems (DFS), multiple machines are used to provide the file system's facility. Different file systems often employ different conceptual models. The models based on structure and mobility are commonly used for the modeling of files.



There are two types of file models:

- Unstructured and Structured Files
- Mutable and Immutable Files

Based on the structure criteria, file models are of two types:

1. Unstructured Files: It is the simplest and most commonly used model. A file is a collection of an unstructured sequence of data in the unstructured model. There is no substructure associated with it. The data and structure of each file available in the file system is an uninterpreted sequence of bytes as it relies on the application used like UNIX or DOS. Most modern OS prefers to use the unstructured file model instead of the structured file model because of sharing of files by different applications. It follows no structure so different applications can interpret in different ways.

2. Structured Files: The rarely used file model now is the Structured file model. Here in the structured file model, the file system sees a file consisting of a collection of a sequence of records in order. Files exhibit different types, different sizes, and different properties. It can also be possible that records of different files belonging to the same file system are of variant sizes. Files possess different properties despite they belong to the same file system. The smallest unit of data that can be retrieved is termed a record. The read or write operations are performed on a set of records. In a structured files system, there are various "File Attributes" available, which describe the file. Each attribute consists of a name with its value. File attributes rely on the file system used. It contains information regarding files, file size, file owner, date of last modification, date of file creation, access permission, and date of last access. The Directory Service facility is used to maintain file attributes because of the varying access permissions.

The structured files further consist of two types:

- Files with Non-Indexed records: In files with non-indexed records, the retrieving of records is performed concerning a position in the file. For example, the third record from the beginning, the third record from the last/end.
- Files with Indexed records: In files with indexed records, one or more key fields exist in each record, each of which can be addressed by providing its value. To locate records fast, a file is maintained as a B-tree or other equivalent data structure or hash table.

Based on the modifiability criteria, file models are of two types:

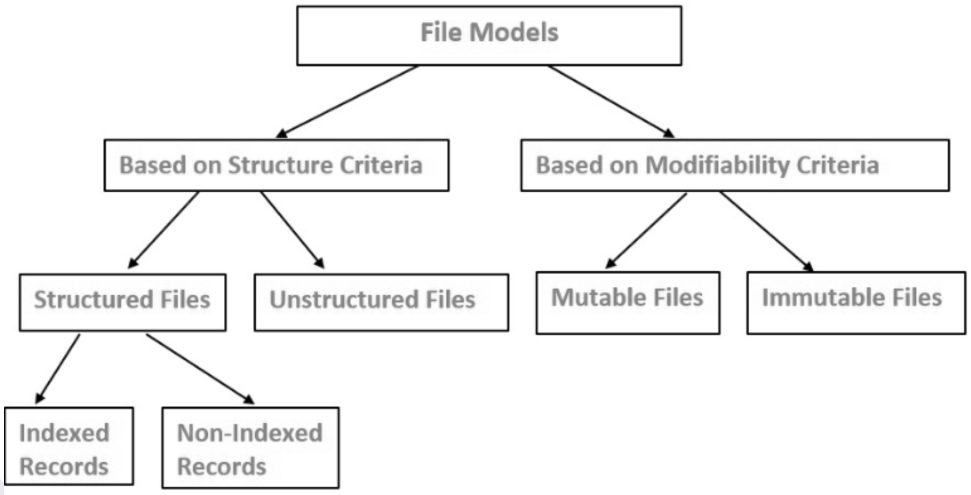
3. Mutable Files: The mutable file model is used by the existing OS. The existing contents of a file get overwritten by the new contents after file updating. As the same file gets updated again and again after writing new contents, a file is described as a single sequence of records.

4. Immutable Files: Cedar File System uses the Immutable file model. In the immutable file model, the file cannot be changed once it has been created. The file can only be deleted after its creation. To implement file updates, multiple versions are created of the same file. Every time a new version of the file is created when a file is updated. There is consistent sharing in this file model because of the sharing of only immutable files. Distributed Systems support caching and replication schemes and hence, overcome the limitation to maintain consistency of multiple copies. Drawbacks of using the Immutable

file model- increase in space utilization and increase in disk allocation activity. CFS employs the "Keep" parameter to maintain the no. of the current version of the file. When the value of the parameter is 1 then it causes the creation of a new file version. The existing version gets deleted and the disk space is reused for another one. When the value of the parameter is greater than 1 then that refers to the existence of multiple versions of a file. The specific version of a file can be accessed by mentioning its full name. In case the version number is not mentioned then CFS uses the lowest version number for the implementation of operations like the "delete" operation and the highest version number for the other operations like the "open" operation.

- **File Accessing models**

Different file systems utilize different conceptual models of a file. The two most usually involved standards for file modeling are structure and modifiability. File models in view of these standards are described below.



File Accessing Models:

The file accessing model basically to depends on

- The unit of data access/Transfer
- The method utilized for accessing to remote files

Based on the unit of data access, following file access models may be utilized to get to the particular file.

1. File-level transfer model: In file level transfer model, the all out document is moved while a particular action requires the document information to be sent the whole way through the circulated registering network among client and server. This model has better versatility and is proficient.

2. Block-level transfer model: In the block-level transfer model, record information travels through the association among clients and a server is accomplished in units of document blocks. Thus, the unit of information moved in the block-level transfer model is document blocks. The block-level transfer model might be used in dispersed figuring climates containing a few diskless workstations.

3. Byte-level transfer model: In the byte-level transfer model, record information moves the association among clients and a server is accomplished in units of bytes. In this way, the unit of information moved in the byte-level exchange model is bytes. The byte-level exchange model offers more noteworthy versatility in contrast with the other record move models since it licenses recuperation and limit of a conflicting progressive sub range of a document. The significant hindrance to the byte-level exchange model is the trouble in store organization because of the variable-length information for different access requests.

4. Record-level transfer model: The record-level file transfer model might be used in the document models where the document contents are organized as records. In the record-level exchange model, document information travels through the organization among clients and a server is accomplished in units of records. The unit of information moved in the record-level transfer model is record.

The Method Utilizes for Accessing Remote Files:

A distributed file system might utilize one of the following models to service a client's file access request when the accessed to file is remote:

1. Remote service model: Handling of a client's request is performed at the server's hub. Thus, the client's solicitation for record access is passed across the organization as a message on to the server, the server machine plays out the entrance demand, and the

result is shipped off the client. Need to restrict the amount of messages sent and the vertical per message.

- Remote access is taken care of across the organization so it is all the slower.
- Increase server weight and organization traffic. Execution undermined.
- Transmission of a series of responses to explicit solicitation prompts higher organization overhead.
- For staying aware of consistency correspondence among client and server is there to have a specialist copy predictable with clients put away data.
- Remote assistance better when essential memory is close to nothing.
- It is only an augmentation of the neighborhood record system interface across the network.

2. Data-caching model: This model attempts to decrease the organization traffic of the past model by getting the data from the server center. This exploits the region part of the found in record gets to. A replacement methodology, for instance, LRU is used to keep the store size restricted.

- Remote access can be served locally so that access can be quicker.
- Network traffic, server load is reduced. Further develops versatility.
- Network overhead is less when transmission of huge amounts of information in comparison to remote service.
- For keeping up with consistency, if less writes then better performance in maintaining consistency ,if more frequent writes then poor performance.
- Caching is better for machines with disk or large main memory.
- Lower level machine interface is different from the upper level UI(user interface).

• File-Caching Schemes

A file-caching scheme for a distributed file system contributes to its scalability and reliability as it is possible to cache remotely located data on a client node. Every distributed file system uses some form of file caching.

The following can be used:

1.Cache Location

Cache location is the place where the cached data is stored. There can be three possible cache locations

i.Servers main memory:

A cache located in the server's main memory eliminates the disk access cost on a cache hit which increases performance compared to no caching.

The reason for keeping locating cache in server's main memory-

- Easy to implement
- Totally transparent to clients
- Easy to keep the original file and the cached data consistent.

ii.Clients disk:

If cache is located in clients disk it eliminates network access cost but requires disk access cost on a cache hit. This is slower than having the cache in servers main memory. Having the cache in the server's main memory is also simpler.

Advantages:

- Provides reliability.
- Large storage capacity.
- Contributes to scalability and reliability.

Disadvantages:

- Does not work if the system is to support diskless workstations.
- Access time is considerably large.

iii.Clients main memory

A cache located in a client's main memory eliminates both network access cost and disk access cost. This technique is not preferred to a client's disk cache when large cache size and increased reliability of cached data are desired.

Advantages:

- Maximum performance gain.
- Permits workstations to be diskless.

- Contributes to reliability and scalability.

2.Modification propagation

When the cache is located on a client's nodes, a file's data may simultaneously be cached on multiple nodes. It is possible for caches to become inconsistent when the file data is changed by one of the clients and the corresponding data cached at other nodes are not changed or discarded.

The modification propagation scheme used has a critical effect on the systems performance and reliability.

Techniques used include –

i.Write-through scheme

When a cache entry is modified, the new value is immediately sent to the server for updating the master copy of the file.

Advantage:

- High degree of reliability and suitability for UNIX-like semantics.
- The risk of updated data getting lost in the event of a client crash is low.

Disadvantage:

- Poor Write performance.

ii.Delayed-write scheme

To reduce network traffic for writes the delayed-write scheme is used. New data value is only written to the cache when an entry is modified and all updated cache entries are sent to the server at a later time.

There are three commonly used delayed-write approaches:

- Write on ejection from cache:

Modified data in cache is sent to the server only when the cache-replacement policy has decided to eject it from the client's cache. This can result in good performance but there can be a reliability problem since some server data may be outdated for a long time.

- Periodic write:

The cache is scanned periodically and any cached data that has been modified since the last scan is sent to the server.

- Write on close:

Modification to cached data is sent to the server when the client closes the file. This does not help much in reducing network traffic for those files that are open for very short periods or are rarely modified.

Advantages:

- Write accesses complete more quickly that result in a performance gain.
- Disadvantage:
- Reliability can be a problem.

3.Cache validation schemes

The modification propagation policy only specifies when the master copy of a file on the server node is updated upon modification of a cache entry. It does not tell anything about when the file data residing in the cache of other nodes is updated. A file data may simultaneously reside in the cache of multiple nodes. A client's cache entry becomes stale as soon as some other client modifies the data corresponding to the cache entry in the master copy of the file on the server. It becomes necessary to verify if the data cached at a client node is consistent with the master copy. If not, the cached data must be invalidated and the updated version of the data must be fetched again from the server.

There are two approaches to verify the validity of cached data:

i.Client-initiated approach

The client contacts the server and checks whether its locally cached data is consistent with the master copy.

- Checking before every access- This defeats the purpose of caching because the server needs to be contacted on every access.
- Periodic checking- A check is initiated every fixed interval of time.
- Check on file open- Cache entry is validated on a file open operation.

ii.Server-initiated approach

A client informs the file server when opening a file, indicating whether a file is being opened for reading, writing, or both. The file server keeps a record of which client has which file open and in what mode. The server monitors file usage modes being used by different clients and reacts whenever it detects a potential for inconsistency. E.g. if a file is open for reading, other clients may be allowed to open it for reading, but opening it for writing cannot be allowed. So also, a new client cannot open a file in any mode if the file is open for writing.

When a client closes a file, it sends intimation to the server along with any modifications made to the file. Then the server updates its record of which client has which file open in which mode.

When a new client makes a request to open an already open file and if the server finds that the new open mode conflicts with the already open mode, the server can deny the request, queue the request, or disable caching by asking all clients having the file open to remove that file from their caches.