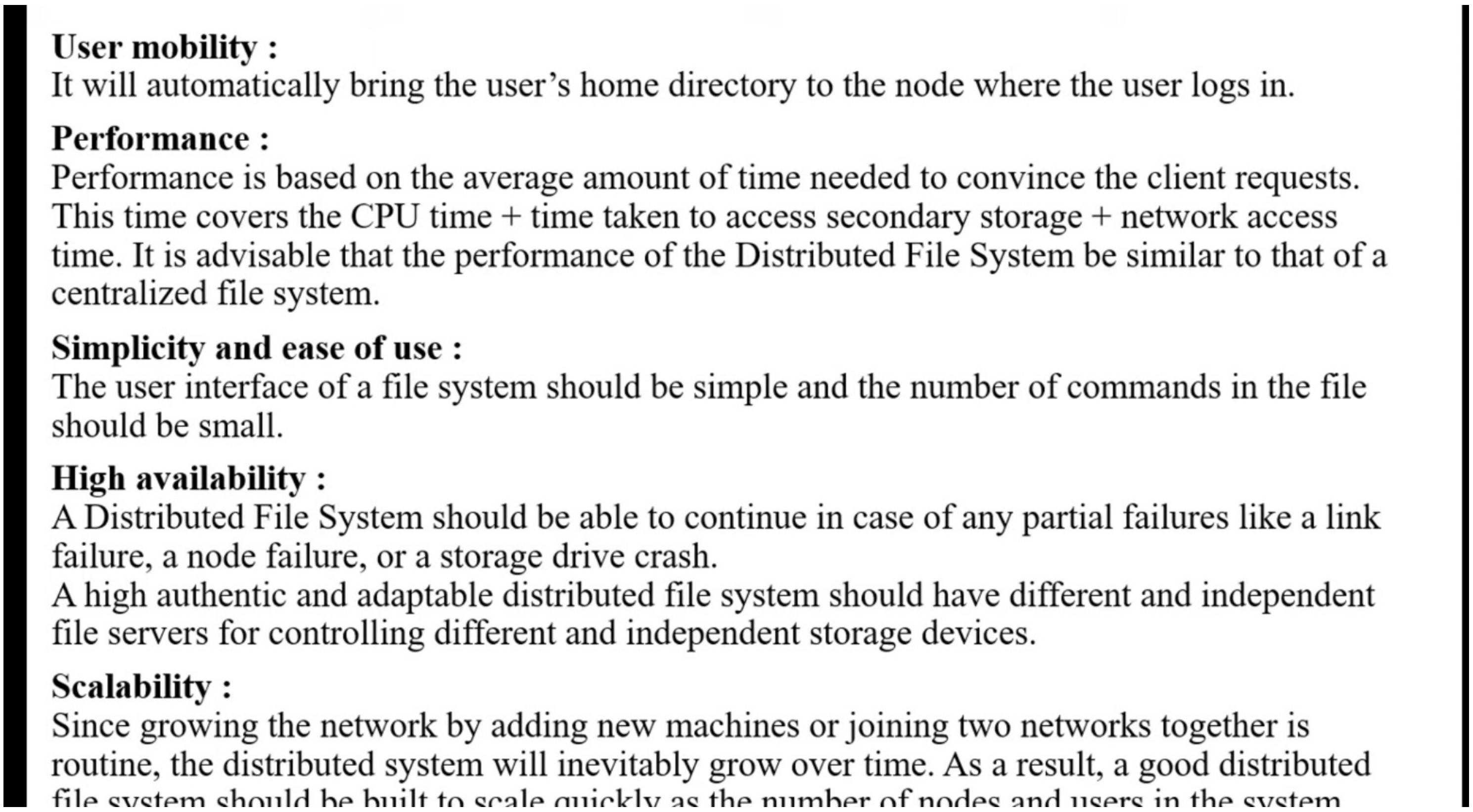
Features of Distributed File Systems:

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.

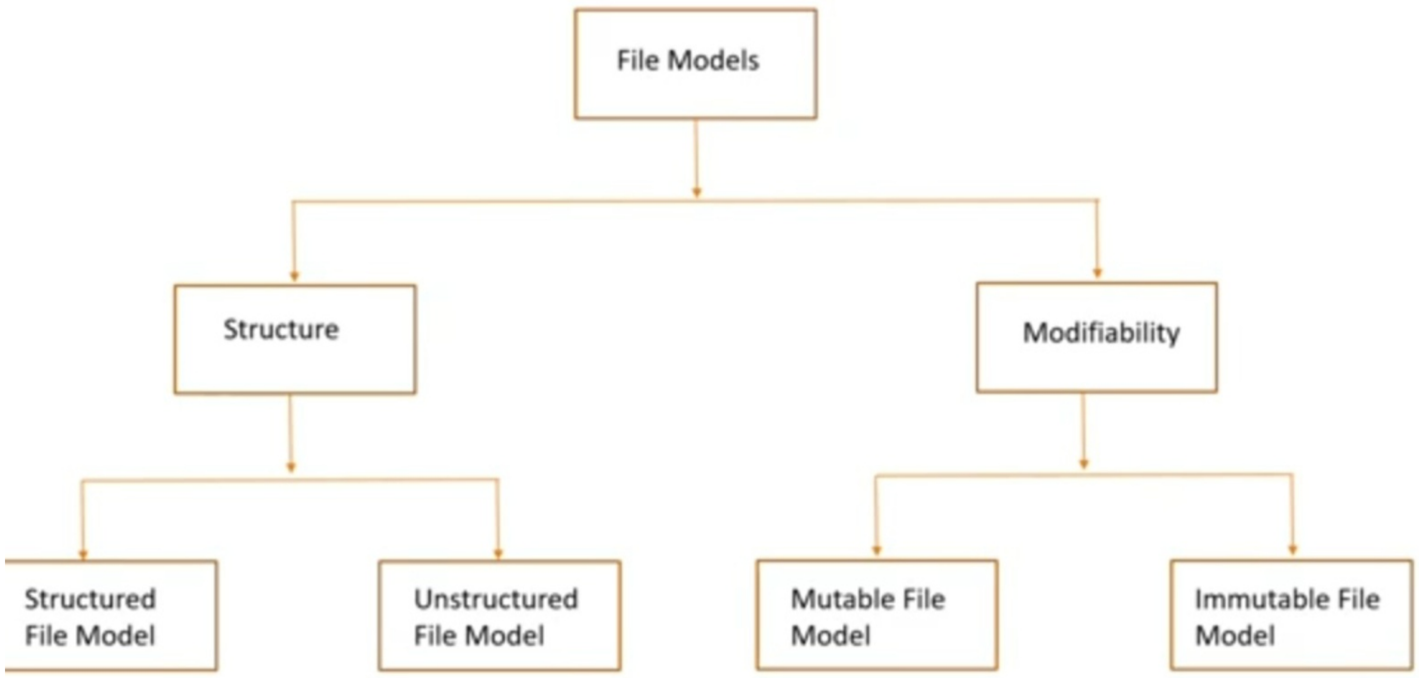
High availability :
A Distributed File System should be able to continue in case of any partial failures like a link failure, a node failure, or a storage drive crash.
A high authentic and adaptable distributed file system should have different and independent file servers for controlling different and independent storage devices.

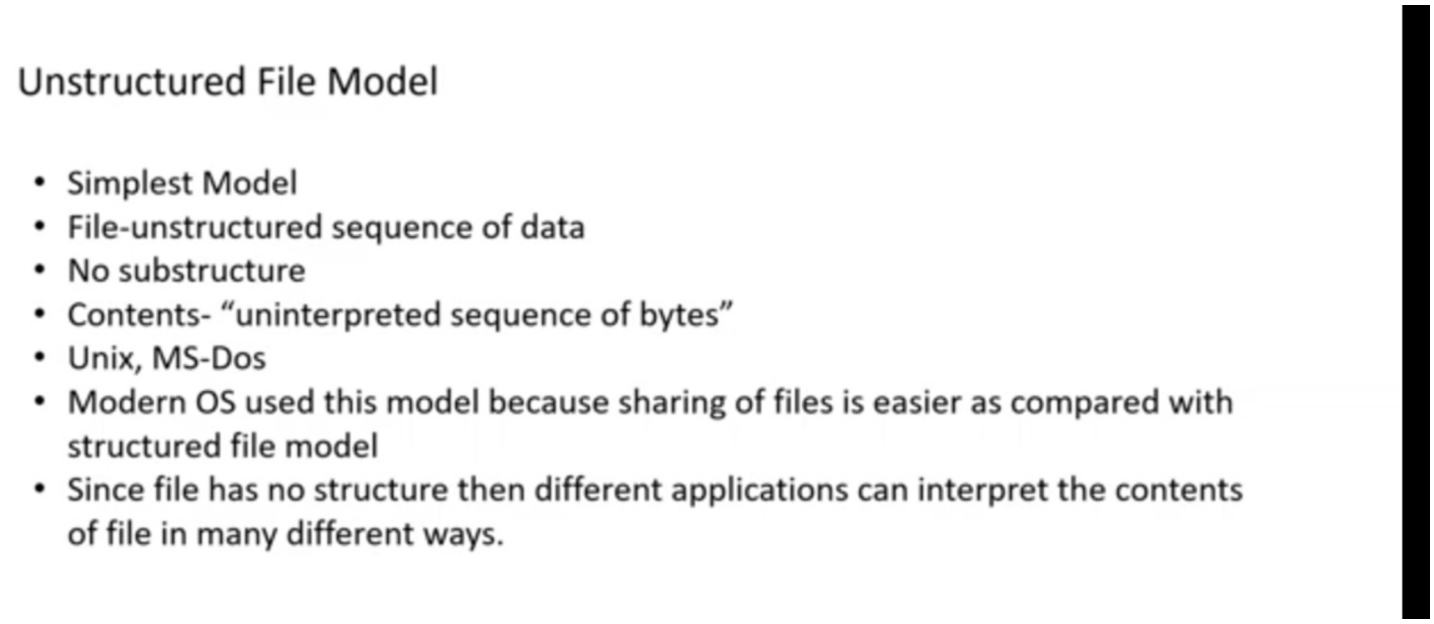
Scalability :
Since growing the network by adding new machines or joining two networks together is routine, the distributed system will inevitably grow over time. As a result, a good distributed file system should be built to scale quickly as the number of nodes and users in the system

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.

Heterogeneity :
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.


File Models
Structure
Modifiability
Structured File Model
Unstructured File Model
Mutable File Model
Immutable File Model



Unstructured File Model

- Simplest Model
- File-unstructured sequence of data
- No substructure
- Contents- “uninterpreted sequence of bytes”
- Unix, MS-Dos
- Modern OS used this model because sharing of files is easier as compared with structured file model
- Since file has no structure then different applications can interpret the contents of file in many different ways.

Structured File Model

- Rarely used
- File-ordered sequence of records
- Files-different types,different size and different properties.
- Record-smallest unit of data that can be accessed
- Two categories
  - Files with non-Indexed records
  - Files with Indexed records

- Files with non-indexed records
  - File records is accessed by specifying it's position within file
  - Ex.Fifth record from beginning,Second record from end

- Files with indexed records
  - Records have one or more key fields that can be addressed by specifying values
  - File is maintained as B-tree or other suitable data structure or hash table to locate records quickly.

Mutable File Model

- Used by most existing os
- Update performed on files overwrites on old contents to produce new contents
- File is represented as a single stored sequence

File Models

Different file system used different conceptual models

The two most commonly used for a file modelling are

Structure And unstructured

Mutable and immutable files

Unstructured and Structured files: In structured files (rarely used now) a file appears to the file server as an ordered sequence of records. Records of different files of the same file system can be of different sizes. In the unstructured model, a file is an unstructured sequence of bytes. The interpretation of the meaning and structure of the data stored in the files is up to the application (e.g. UNIX and MS-DOS). Most modern operating systems use the unstructured file model.

Mutable and immutable files: Based on the modifiability criteria, files are of two types, mutable and immutable. Most existing operating systems use the mutable file model. An update performed on a file overwrites its old contents to produce the new contents.

In the immutable model, rather than updating the same file, a new version of the file is created each time a change is made to the file contents and the old version is retained unchanged. The problems in this model are increased use of disk space and increased disk activity.

File Accessing Models.

The specific client's request for accessing a particular file is serviced on the basis of the file accessing model used by the distributed file system.

The file accessing model basically depends on 1) the unit of data access and 2) the method used for accessing remote files.

On the basis of the unit of data access, following file access models might be used in order to access the specific file.

- File-level transfer model
- Block-level transfer model
- Byte-level transfer model
- Record-level transfer model

1.File-level transfer model: In file-level transfer model, the complete file is moved while a particular operation necessitates the file data to be transmitted all the way through the distributed computing network amongst client and server. This model has better scalability and is efficient.

2.Block-level transfer model: In block-level transfer model, file data transfers through the network amongst client and a server is accomplished in units of file blocks. In short, the unit of data transfer in block-level transfer model is file blocks. The block-level transfer model might be used in distributed computing environment comprising several diskless workstations.

3.Byte-level transfer model: In byte-level transfer model, file data transfers the network amongst client and a server is accomplished in units of bytes. In short, the unit of data transfer in byte-level transfer model is bytes. The byte-level transfer model offers more flexibility in comparison to the other file transfer models since, it allows retrieval and storage of an arbitrary sequential subrange of a file. The major disadvantage of byte-level transfer model is the trouble in cache management because of the variable-length data for different access requests.

4.Record-level transfer model: The record-level file transfer model might be used in the file models where the file contents are structured in the form of records. In record-level transfer model, file data transfers through the network amongst client and a server is accomplished in units of records. The unit of data transfer in record-level transfer model is record.

File Accessing models

↓

Client's request to access a file is serviced depends on the file accessing model → used by file system.

↓

file accessing models of a distributed file system depends on two factors

↓

Method used for accessing remote file

↓

→ Remote service model

↓

Client → Server

↓

Work done at server side

→ Data caching model → remote file access

↑

handling by DFS

reduce request traffic. (LRU) method used to reduce

↓

Unit of data access.

↓

file level transfer

Block level transfer

Byte-level transfer

Record-level transfer