Distributed File System

**What is DFS (Distributed File System)?**

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 allows 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 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 achieves 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.

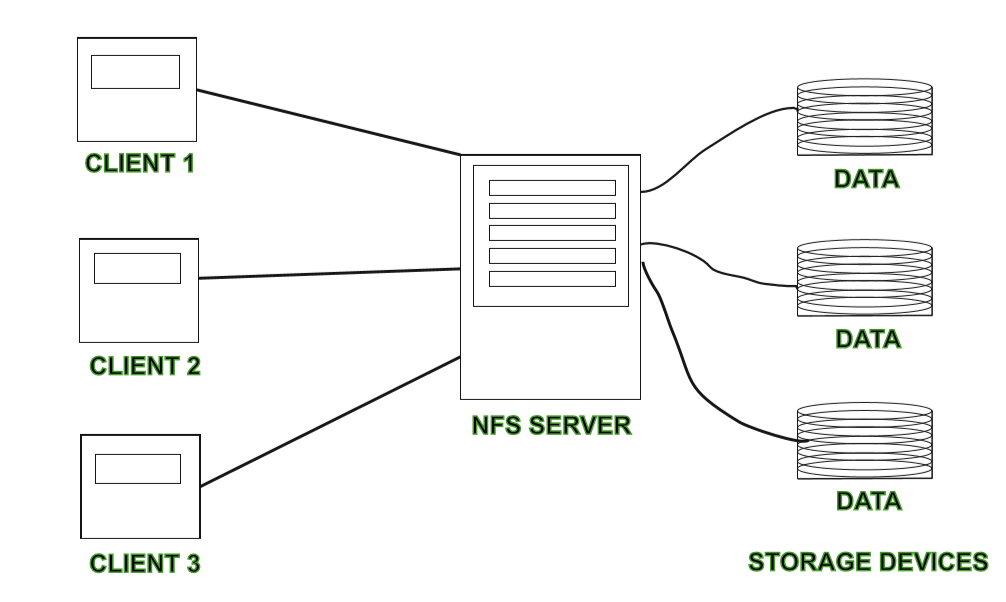
#### Applications :

* **NFS –**  
  NFS stands for Network File System. It is a client-server architecture that allows a computer user to view, store, and update files remotely. The protocol of NFS is one of the several distributed file system standards for Network-Attached Storage (NAS).
* **CIFS –**  
  CIFS stands for Common Internet File System. CIFS is an accent of SMB. That is, CIFS is an application of SIMB protocol, designed by Microsoft.
* **SMB –**  
  SMB stands for Server Message Block. It is a protocol for sharing a file and was invented by IMB. The SMB protocol was created to allow computers to perform read and write operations on files to a remote host over a Local Area Network (LAN). The directories present in the remote host can be accessed via SMB and are called as “shares”.
* **Hadoop –**  
  Hadoop is a group of open-source software services. It gives a software framework for distributed storage and operating of big data using the MapReduce programming model. The core of Hadoop contains a storage part, known as Hadoop Distributed File System (HDFS), and an operating part which is a MapReduce programming model.
* **NetWare –**  
  NetWare is an abandon computer network operating system developed by Novell, Inc. It primarily used combined multitasking to run different services on a personal computer, using the IPX network protocol.

#### Working of DFS :

There are two ways in which DFS can be implemented:

* **Standalone DFS namespace –**  
  It allows only for those DFS roots that exist on the local computer and are not using Active Directory. A Standalone DFS can only be acquired on those computers on which it is created. It does not provide any fault liberation and cannot be linked to any other DFS. Standalone DFS roots are rarely come across because of their limited advantage.
* **Domain-based DFS namespace –**  
  It stores the configuration of DFS in Active Directory, creating the DFS namespace root accessible at **\\<domainname>\<dfsroot>** or **\\<FQDN>\<dfsroot>**



#### Advantages :

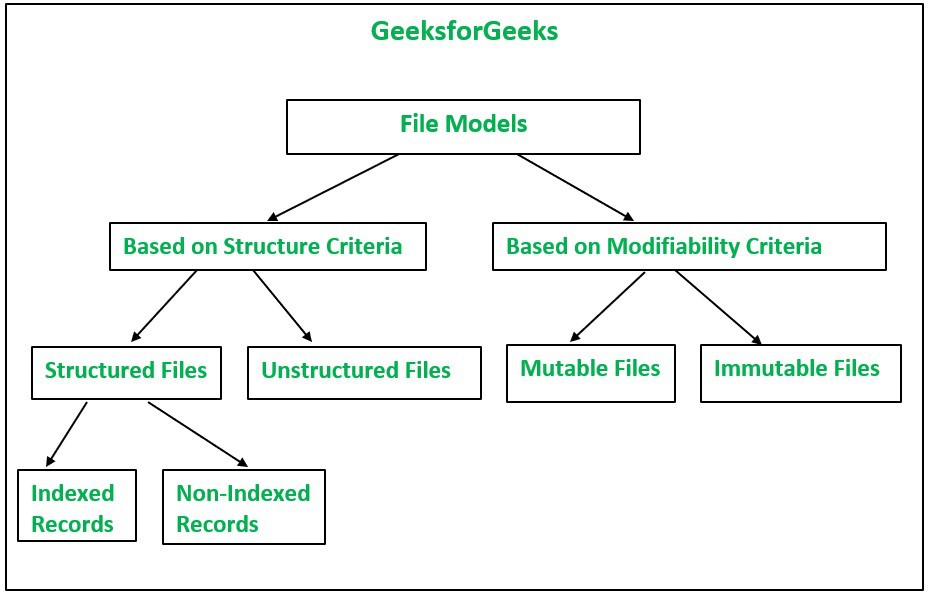
* DFS allows multiple user to access or store the data.
* It allows the data to be share remotely.
* It improved the availability of file, access time, and network efficiency.
* Improved the capacity to change the size of the data and also improves the ability to exchange the data.
* Distributed File System provides transparency of data even if server or disk fails.

#### Disadvantages :

* In Distributed File System nodes and connections needs to be secured therefore we can say that security is at stake.
* There is a possibility of lose of messages and data in the network while movement from one node to another.
* Database connection in case of Distributed File System is complicated.
* Also handling of the database is not easy in Distributed File System as compared to a single user system.
* There are chances that overloading will take place if all nodes tries to send data at once.

**File Models in Distributed System**

In this article, we will go through the concept of File Models in Distributed Systems. 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 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.

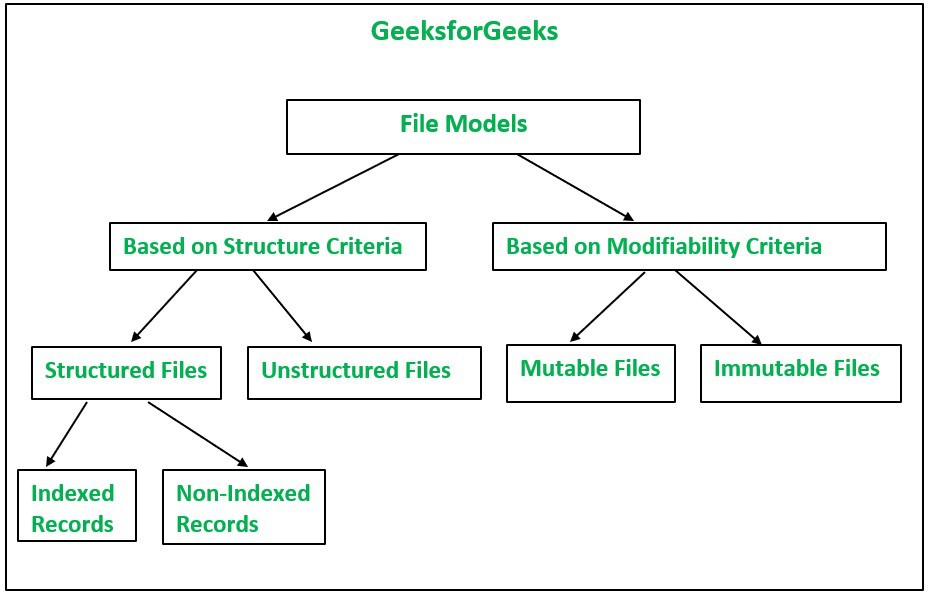
Based on the ***modifiability criteria***, file models are of ***two*** types:

**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 so 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 in Distributed System**

In Distributed File Systems (DFS), multiple machines are used to provide the file system’s facility. Different file system 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 client and a server is accomplished in units of document blocks. Thus, the unit of information move in block-level transfer model is document blocks. The block-level transfer model might be used in dispersed figuring climate containing a few diskless workstations.

**3. Byte-level transfer model:**In the byte-level transfer model, record information moves the association among client and a server is accomplished in units of bytes. In this way, the unit of information move in 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 rang 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 record-level exchange model, document information travels through the organization among client and a server is accomplished in units of records. The unit of information move in 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. Thusly, 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 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 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 got 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 over head is less when transmission of huge 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 upper level UI(user interface).

**Benefit of Data-caching model over the Remote service model:**

The data -catching model offers the opportunity for expanded execution and greater system versatility since it diminishes network traffic, conflict for the network, and conflict for the document servers. Hence almost all distributed file systems implement some form of caching.

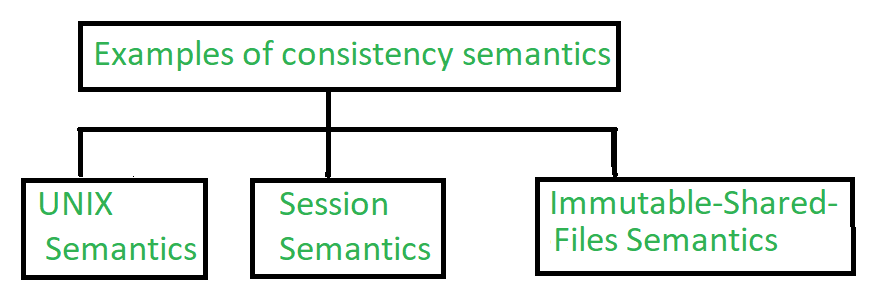
**Example:**NFS utilizes the remote service model but adds caching for better execution.

**Consistency Semantics for file sharing**

**Consistency Semantics** is concept which is used by users to check file systems which are supporting file sharing in their systems. Basically, it is specification to check that how in a single system multiple users are getting access to same file and at same time. They are used to check various things in files, like when will modification by some user in some file is noticeable to others.

Consistency Semantics is concept which is in a direct relation with concept named process synchronization algorithms. But [process synchronization](https://www.geeksforgeeks.org/introduction-of-process-synchronization/) algorithms are not used in case of file I/O because of several issues like great latency, slower rate of transfer of disk and network.

**Example :** When an atomic transaction to remote disk is performed by user, it involves network communications, disks read and write or both. System which is completing their task with full set of functionalities, had a poor performance. In Andrew file system, successful implementation of sharing semantics is found.

To access same file by user process is always enclosed between open() and close() operations. When there are series of access take place for same file, then it makes up a file session.  **1.**

**1.UNIX Semantics :**

The file systems in UNIX uses following consistency semantics –

* The file which user is going to write will be visible to all users who are sharing that file at that time.
* There is one mode in UNIX semantics to share files via sharing pointer of current location. But it will affect all other sharing users.

In this, a file which is shared is associated with a single physical image that is accessed as an exclusive resources. This single image causes delays in user processes.

**2. Session Semantics :**

The file system in Andrew uses following consistency semantics.

* The file which user is going to be write will not visible to all users who are sharing that file at that time.
* After closing file, changes done to that file by user are only visible only in sessions starting later. If changes file is already open by other user, then changes will not be visible to that user.

In this, a file which is shared is associated with a several images and there is no delay in this because it allows multiple users to perform both read and write accesses concurrently on their images.

**3. Immutable-Shared-Files Semantics :**

There seems a unique approach immutable shared files. In this, user are not allowed to modify file, which is declared as shared by its creator. An Immutable file has two properties which are as follows –

* Its name may not be reused.
* Its content may not be altered.

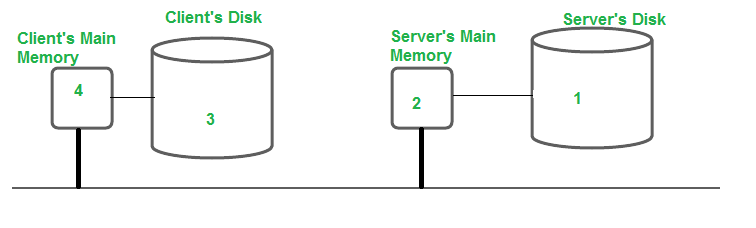
In this file system, content of file are fixed. The implementation of semantics in a distributed system is simple, because sharing is disciplined.

**File Caching in Distributed File Systems**

File caching enhances I/O performance because previously read files are kept in the main memory. Because the files are available locally, the network transfer is zeroed when requests for these files are repeated. Performance improvement of the file system is based on the locality of the file access pattern. [Caching](https://www.geeksforgeeks.org/difference-between-buffering-and-caching-in-os/) also helps in reliability and scalability.

The majority of today’s distributed file systems employ some form of caching. File caching schemes are determined by a number of criteria, including cached data granularity, cache size (large/ small/ fixed/ dynamic), replacement policy, cache location, modification propagation mechanisms, and cache validation.

**Cache Location:** The file might be kept in the disc or main memory of the client or the server in a client-server system with memory and disk.



**Server’s Disk:**It is always the original location where the file is saved. There is enough space here in case that file is modified and becomes longer. Additionally, the file is visible to all clients.

**Advantages:** There are no consistency issues because each file has only one copy. When a client wants to read a file, two transfers are required: from the server’s disk to the main memory, and from the client’s main memory to the server’s disk.

**Disadvantages:**

* It’s possible that both of these transfers will take some time. One part of the transfer time can be avoided by caching the file in the server’s main memory to boost performance.
* Because main memory is limited, an algorithm will be required to determine which files or parts of files should be maintained in the cache. This algorithm will be based on two factors: the cache unit and the replacement mechanism to apply when the cache is full.

**Server’s Main Memory:** The question is whether to cache the complete file or only the disk blocks when the file is cached in the server’s main memory. If the full file is cached, it can be stored in contiguous locations, and high-speed transmission results in a good performance. Disk block caching makes the cache and disc space more efficient.  
Standard caching techniques are employed to overcome the latter issue. When compared to memory references, cache references are quite rare. The oldest block can be picked for eviction in [LRU (Least Recently Used)](https://www.geeksforgeeks.org/lru-cache-implementation/). The cache copy can be discarded if there is an up-to-date copy on the disk. The cache data can also be written to the disk. Clients can easily and transparently access a cached file in the server’s main memory. The server can easily keep disks and main memory copies of the file consistent. Only one copy of the file exists in the system, according to the client.

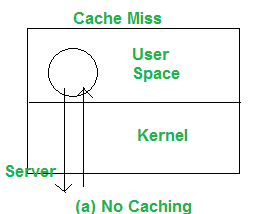
**Client’s disk:** The data can also be saved on the client’s hard drive. Although network transfer is reduced, in the event of a cache hit, the disk must be accessed. Because the changed data will be available in the event of data loss or a crash, this technique improves reliability. The information can then be recovered from the client’s hard drive.    
Even if the client is disconnected from the server, the file can still be accessed. Because access to the disk may be handled locally, there is no need to contact the server, this enhances scalability and dependability.

**Advantages:**

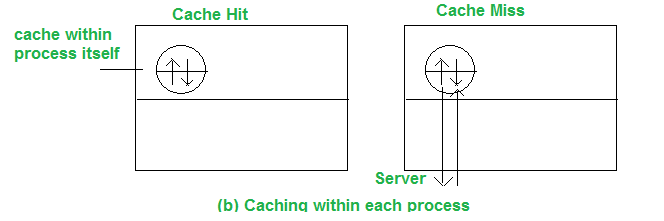
* Reliability increased as data can be recovered in case of data loss.
* The client’s disk has a significantly larger storage capacity than the client’s primary memory. It is possible to cache more data, resulting in the highest cache-hit ratio. The majority of distributed file systems employ a file-level data transfer architecture, in which the entire file is cached.
* Scalability is increased as access to the disk can be handled locally.

**Disadvantages:**

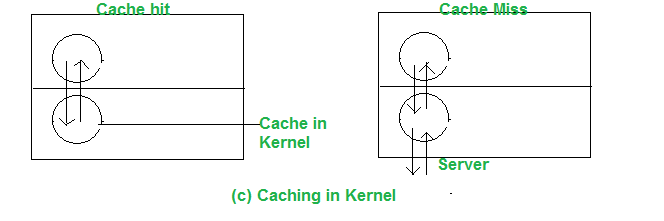
* The sole drawback is that disc caching is incompatible with disk-less workstations. Every cache requires disk access, resulting in a considerable increase in the response time. It must be decided whether to cache in the server’s main memory or on the client’s disc.
* Although server caching eliminates the need for disk access, network transfer is still required. Caching data on the client-side is a solution for reducing network transfer time. Whether the system should use the client’s main memory or the disk, depends on whether the system needs to save space or improve performance.
* The access is slow if the disk has more space. The main memory of the server may be able to provide a file faster than the client’s disc. Caching can be done on the client’s disc if the file size is very high. The below figure shows the simplest way, i.e., avoid caching.

[](https://media.geeksforgeeks.org/wp-content/uploads/20210710162132/t1.png)

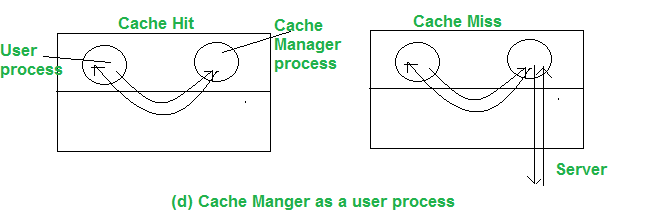
**Client’s Main Memory:**Once it is agreed that the files should be cached in the client’s memory, caching can take place in the user process’s address space, the kernel, or a cache manager as a user process.  
The second alternative is to cache the files in the address space of each user process, as shown:

[](https://media.geeksforgeeks.org/wp-content/uploads/20210710162630/t3.png)

The system-call library is in charge of the cache. The files are opened, closed, read, and written during the process execution. The library saves the most frequently used files so that they can be re-used if necessary. The updated files are returned to the server once the operation has been completed. When individual processes open and close files regularly, this technique works well.   
It’s fine for database managers, but not for programmers working in circumstances where the files might not be accessed again.   
The file can be cached in the kernel instead of the user’s process address space, as shown. This technique, however, necessitates many systems calls to access the file for each cache hit.

[](https://media.geeksforgeeks.org/wp-content/uploads/20210710163222/t4.png)

A separate user-level cache manager can be used to cache the files. As a result, the kernel no longer has to maintain the file system code, and it becomes more isolated and flexible. The kernel can decide on the allocation of memory space for the program vs. cache on run time. The kernel can store some of the cached files in the disk if the cache manager runs in virtual memory, and the blocks are brought to the main memory on cache hit.

[](https://media.geeksforgeeks.org/wp-content/uploads/20210710164057/t6.png)

**Advantages:**

* This technique is more isolated and flexible (as the kernel no longer has to maintain the file system code)
* When individual processes open and close files regularly, the access time decreases. So, Gain in performance is maximum.
* Allows for diskless workstations.
* Contributes to the scalability and reliability of the system.

**Disadvantages:**

* A separate user-level cache manager is required.
* Client caching principles have no value with virtual memory, although the cache manager can lock some frequently requested pages.

### ****Cache Consistency – Cache Update Policy:****

When the cache is located on the client’s node, numerous users can access the same data or file at the same time in a file system. If all caches contain the same most current data, they are considered to be consistent. It’s possible that the data will become inconsistent if some users modify the file. The distributed system that uses a DFS must keep its data copies consistent.  
Depending on when to propagate changes to the server and how to validate the authenticity of cache data, many consistency strategies are provided. Write-through, write-on-close, and centralized control are the three types.  
When the cache is located on the client’s node & one user writes data to cache, it must also be visible to the other users as well. The written policy determines that when the writing is performed.

There are four cache update policies:

* **Write-Through:**When a new user edits a cache entry in this method, it is immediately written to the server. Any procedure that requires a file from the server will now always receive the most up-to-date information. Consider the following scenario: the client process reads the file, caches it, and then exits the process. Another client modifies the same file and sends the change to the server a short time later.  
  If a process is started on the first machine with the cached copy of the file, it will obtain an outdated copy. To avoid this, compare the time of modification of both copies, the cached copy on the client’s machine and the uploaded copy on the server, to validate the file with the server.
* **Delayed Write:**To reduce continuous network traffic, write all updates to the server periodically or batch them together. It’s known as ‘delayed-write.’ This method enhances performance by allowing for a single bulk write operation rather than several tiny writes. The temporary file is not stored on the file server in this case.
* **Write on close:**One step forward is to only write the file back to the server once it has been closed. ‘Write on close’ is the name of the algorithm. The second write overwrites the first if two cached files are written back to back. It’s comparable to what happens when two processes read or write in their own address space and then write back to the server in a single CPU system.
* **Centralized Control:**For tracking purposes, the client sends information about the files it has just opened to the server, which then performs read, write, or both activities. Multiple processes may read from the same file, but once one process has opened the file for writing, all other processes will be denied access. After the server receives notification that the file has been closed, it updates its table, and only then can additional users access the file.

### Cache Validation Scheme:

When a cache’s data is modified, the modification propagation policy tells when the master copy of the file on the server node is updated. It provides no information about when the file data in other nodes’ caches are updated. Data from a file may be stored in the caches of many nodes at the same time.  
When another client alters the data corresponding to the cache item in the master copy of the file on the server, the client’s cache entry becomes outdated. It’s required to check whether the data cached at a client node matches the master copy. If this is not the case, the cached data must be invalidated and a new version of the data must be requested from the server.  
To check cache data’s Validity, 2 schemes are :

1. **Client-initiated Approach:** The client connects to the server and verifies that the data it has in its cache is consistent with the master copy. The checking can be done at different times as-
   * **Verify before each access:**As here the server must be called each time an access is made, this negates the actual purpose of caching data.
   * **Verifying periodically:** Validation is done at a regular predetermined interval
   * **Verify the opening of the file:** The cache entry is checked when a file is opened.
2. **Server-initiated Approach:** When a client opens a file, it informs the file server of the purpose for opening the file – reading, writing, or both. It is then the duty of the file server to keep track of which client is working on which file and in which mode(s). When the server identifies any chance of inconsistency when the file is used by various clients, it reacts.
   * A client notifies the server of the closure, as well as any changes made to the file when it closes a file. The server then updates its database to reflect which clients have which files open in which modes.
   * The server can deny/queue the request, or disable caching by requesting that all clients who have the file open remove it from their caches whenever a new client requests to open a file that is already open and the server discovers any inconsistency which is there/may occur.

**What is Replication in Distributed System?**

In a distributed system data is stored is over different computers in a network. Therefore, we need to make sure that data is readily available for the users. **Availability** of the data is an important factor often accomplished by data replication. ***Replication is the practice of keeping several copies of data in different places.***

### Why do we require replication?

The first and foremost thing is that it makes our system more stable because of node replication. It is good to have replicas of a node in a network due to following reasons:

* If a node stops working, the distributed network will still work fine due to its replicas which will be there. Thus it increases the fault tolerance of the system.
* It also helps in load sharing where loads on a server are shared among different replicas.
* It enhances the availability of the data. If the replicas are created and data is stored near to the consumers, it would be easier and faster to fetch data.

### Types of Replication

* Active Replication
* Passive Replication

### Active Replication:

* The request of the client goes to all the replicas.
* It is to be made sure that every replica receives the client request in the same order else the system will get inconsistent.
* There is no need for coordination because each copy processes the same request in the same sequence.
* All replicas respond to the client’s request.

**Advantages:**

* It is really simple. The codes in active replication are the same throughout.
* It is transparent.
* Even if a node fails, it will be easily handled by replicas of that node.

**Disadvantages:**

* It increases resource consumption. The greater the number of replicas, the greater the memory needed.
* It increases the time complexity. If some change is done on one replica it should also be done in all others.

### Passive Replication:

* The client request goes to the primary replica, also called the main replica.
* There are more replicas that act as backup for the primary replica.
* Primary replica informs all other backup replicas about any modification done.
* The response is returned to the client by a primary replica.
* Periodically primary replica sends some signal to backup replicas to let them know that it is working perfectly fine.
* In case of failure of a primary replica, a backup replica becomes the primary replica.

**Advantages:**

* The resource consumption is less as backup servers only come into play when the primary server fails.
* The time complexity of this is also less as there’s no need for updating in all the nodes replicas, unlike active replication.

**Disadvantages:**

* If some failure occurs, the response time is delayed.

**Atomic Commit Protocol in Distributed System**

### Distributed Transactions:

Distributed transaction refers to the transaction in which multiple servers are involved. Multiple servers are called by a client in Simple Distributed Transaction whereas a server calls another server in Nested Transaction. The execution of a transaction at many sites, must either be committed at all sites or aborted at all sites. But this should not be the case that transaction is committed at one site and aborted at another site. Distributed site systems use distributed commitment rules to ensure atomicity across sites. Atomic commitment is a channel of need for cooperation across a variety of systems.

### Essential Properties of the Distributed Transactions:

The distributed transactions must possess the following ACID properties, like any other transaction. In distributed systems, a transaction manager is used to coordinate the distinct operations and then commit/rollback the transaction as needed.

1. **Atomicity:**All data changes are treated as if they were a single operation. That is, either all of the modifications are made, or none of them are made. The atomicity feature assures that if a debit is successfully made from one account, the matching credit is made to the other account in an application that transfers money from one account to another.
2. **Consistency:** This property implies that when a transaction begins and ends, the state of data is consistent. For example, it ensures that the value remains consistent at the start and end of a transaction in an application that transfers funds from one account to another.
3. **Isolation:** In this property, concurrently running transactions appear to be serialized. For example, the isolation property assures that the transferred funds between two accounts can be seen by another transaction in either one of the accounts, but not both, or neither.
4. **Durability:** Changes to data persist when a transaction completes successfully and are not undone, even if the system fails. It assures that modifications made to each account will not be reversed in an application that transfers money from one account to another.

### Coordination in Distributed Transactions:

At the time of coordination in Distributed Transactions, one of the servers becomes a coordinator, and the rest of the workers become coordinators.

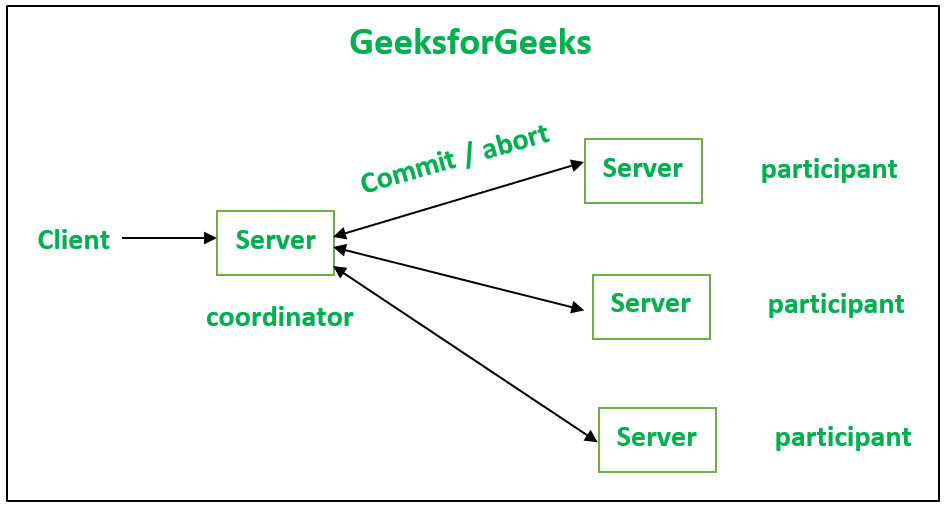
* In a simple transaction, the first server acts as the Coordinator.
* In the nested transaction, the top-level server acts as the Coordinator.
* **Role of Coordinator:**The coordinator keeps track of participating servers, gathers results from workers, and makes a decision to ensure transaction consistency.
* **Role of Workers:**Workers are aware of the coordinator’s existence and in addition, communicate their outcome to the coordinator and then follow the coordinator’s decision.

### Atomic Commit:

The atomic commit procedure should meet the following requirements:

* All participants who make a choice reach the same conclusion
* If any participant decides to commit, then all other participants must have voted yes.
* If all participants vote yes and no failure occurs, then all participants decide to commit.

**Distributed One-Phase Commit:**A one-phase commitment protocol involves a coordinator who communicates with servers and performs each task regularly to inform them to perform or cancel actions i.e. transactions.



**Distributed Two-Phase Commit:**There are two phases for the commit procedure to work:

**Phase 1:** Voting

* A “prepare message” is sent to each participating worker by the coordinator.
* The coordinator must wait until a response whether ready or not ready is received from each worker, or a timeout occurs.
* Workers must wait until the coordinator sends the “prepare” message.
* If a transaction is ready to commit then a “ready” message is sent to the coordinator.
* If a transaction is not ready to commit then a “no” message is sent to the coordinator and resulting in aborting of the transaction.

**Phase 2: Completion of the voting result**

* In this phase, the Coordinator will check about the “ready” message.  If each worker sent a “ready” message then only a “commit” message is sent to each worker; otherwise, send an “abort” message to each worker.
* Now, wait for acknowledgment until it is received from each worker.
* In this phase, Workers wait until the coordinator sends a “commit” or “abort” message; then act according to the message received.
* At last, Workers send an acknowledgment to the Coordinator.

