DISTRIBUTED FILE SYSTEMS

References [Col94], [Gos91], [Sin97], [Tan95]

DISTRIBUTED FILE SYSTEMS

- **Introduction**
- **Services of file servers**
- **Types of file servers**
- **Design issues of DFSs**
- **Architecture of file systems**

SERVICES OF DISTRIBUTED FILE SYSTEMS

DIRECTORY SERVICE

- **Directory and file naming**
- **Access control of files**
- **Placement, location and relocation of files**
- **Directory service operations**

REPLICATED FILE SERVICES

- **Replication transparency**
- **File replication**

TRANSACTION SERVICE

- **Recovery methods**
- **Concurrency control**
- **Transaction service operations**

FILE SERVICE

- **Access to files**
- **File sharing**
- **Fault tolerance**
- **File service operations**

DISK SERVICE

- **File storage**
- **File caching**
- **Disk service operations**

DISTRIBUTED FILE SYSTEMS

Introduction

file:

- **provides permanent storage of information on a secondary storage media,**
- **provides sharing of information, i.e. file is created by one user and then shared with several other users,**

file models:

1. file models based on structure

♦ **unstructured files**	**file is an non-structured sequence of bytes, the interpretation of meaning and the structure of the data stored in the files are defined by the application programs,** **for example: in MS-DOS and UNIX,**
♦ **structured files**	**file is a structured sequence of bytes where record is the smallest unit of file data that can be accessed,**

2. file models based on modifiability

♦ **mutable files**	**an update performed on a file overwrites on its old contents to produce the new contents,**
♦ **inmmutable files**	**a file cannot be modified once it has been created, i.e. a new version of the file is created each time a change is made to the file contents and the old version is kept unchanged,** **using immutable files supports file caching and file replication because it eliminates all problems associated with keeping multiple copies of a file consistent**

file system:

- **abstraction of a storage device that supports file management in computer systems,**

- **subsystem of the OS that performs file management activities for example: organisation, storing, retrieval, naming sharing and protection of files,**

distributed file system

- **DFS is a file system that consists of storage system located at different nodes but appears as a traditional centralised file system for users,**

- **one of the most important services provided by a DS is the distributed file service because other services, for example print service, name service, rely on file service,**

- **file service is the most heavily used service in a DS thus, its functionality and performance are critical,**

- **when a computer needs to access a file stored on a remote computer, it performs the requested file operations without copying files to a local disk,**

properties of distributed file systems:

- **remote information sharing**

  ♦ **DFS allows a file to be transparently accessed by processes of any node of the system not depending of the file's location,**

- **user mobility**

  ♦ **DFS allows a user to work on different nodes at different times without the necessity of physically relocating the secondary storage device,**

- **availability**
  - ♦ **in DFS files should be available for users even after temporary failure one or more nodes of the systems,**
  - ♦ **to provide this capability the DFS keeps multiple copies of a file on different nodes of the system,**
- **diskless workstation**
  - ♦ **DFS allows the use of diskless workstation as a result of transparent remote file accessing capability,**

models of distributed file systems:

<u>**1. integrated model:**</u>

- ♦ **all users has the same view of the system, each computer runs a complete (but highly configurable) set of standard software which provides all the needed facilities,**

<u>**2. client-server model:**</u>

- ♦ **services are designated to servers that run special purpose server software, clients are a priori unknown (and placed on the local or remote machine),**

Services of file servers

file system:

- **the client module integrates services of the file system to provide a single programming interface with operations similar to operations of conventional file systems,**
- **single client module runs in each client integrating and extending the operations of services of file systems using a single API,**

logical components of file systems:

file name	file attributes	data units

file name	**file is assigned with a symbolic name which is mapped to an unique file *id* when locating the physical file,**
file attributes	**contain information about ownership, type, size, timestamps and access authorisation of the file,**
data units	**are organised as either flat structure of bytes or sequence of blocks; or hierarchical structure of indexed records,**

file access approaches:

sequential access	**a file position pointer is maintained by the system for each opened file, to indicate the position of the next data unit to be accessed,**
direct access	**data units are referenced by their blocks numbers implicitly,**

indexed sequential access

data units are accessed by using an index associated with each data block,

basic services of a distributed file system and their tasks:

services	tasks
directory service	**name resolution, add & deletion of files**
replication service	**managing multiple copies of files**
transaction service	**concurrency & replication management**
file service	**read/write files & get/set attributes**
disk service	**device, cache & block management**

1. directory (or name) service

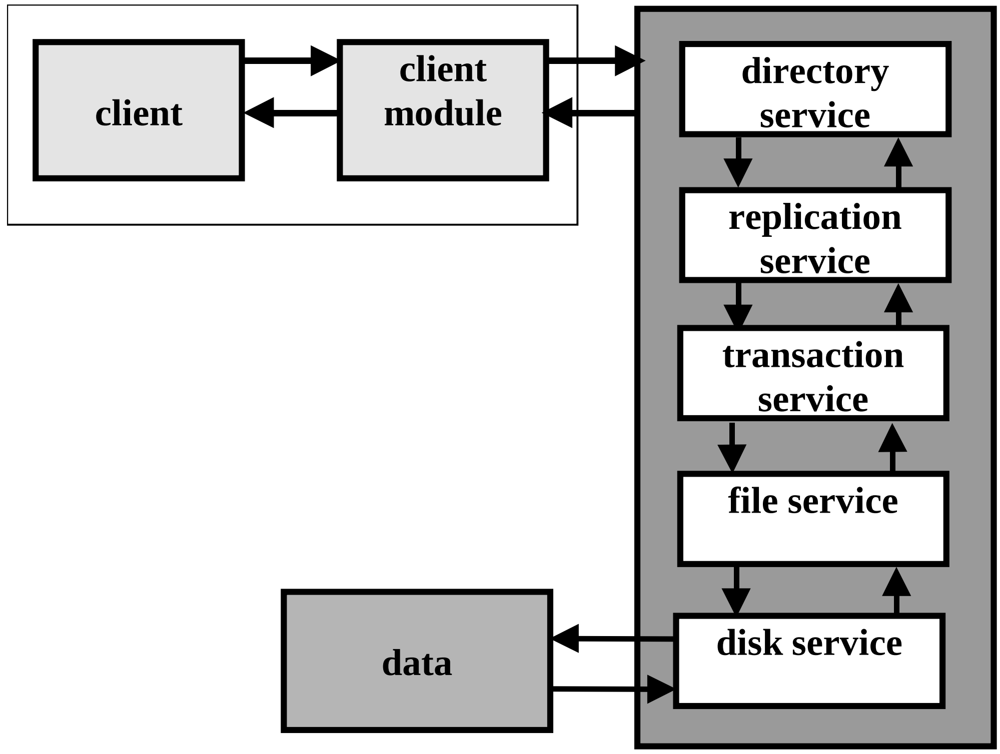
- **provides mapping between user names for files & references to files,**
- **performs directory-related activities for example creation, deletion of directories, adding a new file a directory, deleting a file from a directory, changing the name of a file,**

2. replication + transaction + file (or true file) service

- **supports operations on files for example accessing and modifying data in files, creating and deleting files,**
- **these services manage file accessing mechanism, file sharing semantics, file caching mechanism, file replication mechanism and concurrency control mechanism,**

3. disk (or storage) service

- **manages the allocation and space on a secondary storage device that is used in a file system,**
- **provides a logical view of the storage system by providing operations for storing and retrieving data in them,**



file servers:

- **files are distributed among file servers using file groups and file servers may hold several file groups,**

- **a file group is a collection of files mounted on a server that supports the allocation of files to file servers,**

- **file servers register their services with a registration service, and clients consult with it before mounting, or clients broadcast mounting requests, and file servers respond to clients' request,**

file mounting:

- **file mounting constructs a large file system from various file servers and storage devices by binding different filename spaces,**

- **a mounting operation attaches a remote file system to the client's file system hierarchy at the mount point given to by a specific path name,**

- **once files are mounted, they are accessed by using the concatenated logical path names without referencing either the remote host or local devices,**

file caching:

- **a copy of data stored at a remote file server is brought to the client when referenced by the client,**
- **subsequent access to the data is performed in the client cache in order to reduce access delays due to network latency,**
- **file caching is supported by a cache manager that can be present at both clients and file servers,**
- **cache managers co-ordinate data storage and retrieval operations in order to maintain data consistency,**

Types of file servers

data management methods

data access & locking methods

file services

remote file access

classification aspects

- classification based on data management methods:

1. true file server

- **manages user (or text) names and file directories, i.e. file accessing mechanism, file sharing semantics, file caching mechanism, file replication mechanism, concurrency control mechanism,**

2. storage server

- **manages data as unstructured objects using primitive identifiers,**

- classification based on data access and locking methods:

1. simple file server

- **data access, locking and atomic update uses file as a basic unit of data management,**

2. universal file server:

- **data access is based on sequential subset of files, locking and atomic update uses files as basic units of data management,**

3. file server with database management:

- **data access and locking is based on records and atomic update on files**

- classification based on file services:

1. traditional file service:

- ♦ **it opens, reads and writes files similarly to file systems in traditional computer systems,**

2. robust file service

- ♦ **it provides high reliability by using atomic updates,**

- classification based on remote file access methods:

1. explicit file transfer

- ♦ **a client invokes a file transfer procedure to transfer a remote file before or after its use,**

2. distributed file system

- ♦ **a client refers to files by name, the system locates the server and performs security operation, synchronises access to the file and transfers data,**

Design issues of DFSs

1. wide range of applications

- **file system should support wide range of applications with different requirements to the file system,**

2. matching user requirements

- **a file system should provide only those services to the user needed by him/her, i.e. the user doesn't want to pay for the unnecessary features,**

3. multiplicity

- **multiple clients in several different locations may access files, which are often replicated,**

multiple clients **files can be shared by multiple concurrent users, an update to a file by one process should not have an adverse effect on the correct execution of other processes that are concurrently sharing the file *(concurrency transparency)*,**

a user can log in at any host in the system with a uniform login procedure and perceive a uniform view of the file system, independent of the host *(login transparency)*,

the client processes running on the host have a uniform mechanism to access all files *(access transparency)*,

multiple files **files may be replicated to provide redundancy for availability and permit concurrent accesses for efficiency *(replication transparency)*,**

names used to access files need not contain information about the physical location of the files *(location transparency)*,

files can be moved from one physical location to another without changing their names *(location independence)*,

3. file system semantics

- **semantics should have a simple interface and a small number of commands in order to provide the easy use of the distributed file system,**

4. transparency

location (or naming) transparency

- **a file should be movable in a distributed system without changing it's name and pathname,**
- **every user and process in a distributed system uses the same text name for accessing a file regardless of the location of the object,**

access transparency

- **processes should not need or able to be recognise whether a file is local or remote,**
- **they can access local and remote files in the same way using a single set of operations,**
- **users should be able to access their files from any node in a distributed system (user mobility),**

concurrency transparency

- **changes in a file by one client should not interfere with the operation of other clients simultaneously accessing or changing the same file,**
- **file system must guarantee the integrity of data, changes to files must never be lost, (concurrency control)**

replication transparency

- **a file may have several copies at different sites in a distributed file system,**
- **existence of multiple copies and their locations should be hidden from the clients,**

performance transparency

- **performance of client programs does not depend on load on the service while service changes within a specified range,**

5. availability

- **users should have the same easy access to files, irrespective of their physical location,**
- **the failure of a computer or an element of the network should not affect the whole distributed system,**
- **to manage failure of a server replication is applied to the file system for enhancing the availability of files,**

6. scalability

- **the distributed system should be changed (enlarged) without any loss of performance or service with a wide range of loads and network sizes,**
- **DFS should manage high service load, accommodate growth of the users, and enable simple integration of added resources,**

7. performance

- **users should not be able to recognise the existence of distributed system on the grounds of performance,**
- **it means that the level of file system performance should be at least as good as that of an equivalent time-sharing system,**
- **the performance should be about the same between the user and other computers,**

8. hardware and OS heterogeneity

- **heterogeneous DSs provide the flexibility for the users to use different computer platforms for different applications,**
- **as a result DFS service interfaces should have manage clients and**

servers on different hardware and OS platforms,

- **DFS should be designed to allow the integration of a new type of workstation or storage media in a relatively simple way,**

9. security

- **users should recognise the identity of the other users and server processes,**

- **users should be treated in the same way regardless of the current physical location,**

- **information stored in a file system should be protected against non-authorised access,**

- **safe communication, built-in protection mechanisms should be provided in DFS (e.g. each file usually belongs to its owner who created it, and the system provides different levels of user granularity: owner, group, other)**

10. tolerance to network partitioning

- **when a file service includes replication or caching of files, clients may be affected when a network partition occurs,**

11. crash recovery & fault tolerance

- **servers and user processes must be able to recover easily from computer crashes or network problems by using recovery mechanisms,**

- **automatic backup and recovery is performed against failures in DSs,**

12. user mobility

- **in a DS user should have a flexibility to work on a different nodes at different times,**

- **it is provided by automatically bringing a user's environment at the time of login to the node where the user logs in,**

Architecture of file systems

1. file system in a centralised operating system:

DIRECTORY SERVICE	
directory module	**access control module**
FILE SERVICE	
file module	**file access module**
DISK SERVICE	
block module	**device module**

modules of the file system:

directory module:	**mapping between file names and file Ids,**
access control module:	**checking rights for operations,**
file module:	**identifying particular files,**
file access module:	**file data & attributes reading & writing,**
block module:	**disk blocks accessing and allocation,**
device module:	**disk I/O operations and buffering,**

2. file system in a distributed operating system:

DIRECTORY SERVICE		
directory module	**access control module**	
placement module	**location module**	**relocation module**
REPLICATION SERVICE		
data consistency and multiple copy update module		
TRANSACTION SERVICE		
transaction module	**recovery module**	**concurrency control module**
FILE SERVICE		
file module	**file access module**	**attribute access module**
DISK SERVICE		
block module	**device module**	**caching module**

modules of the file system:

placement module: **selects a server to hold a newly created file,**

location module: **finds the location of the server managing the referenced file,**

relocation module: **improves the performance by relocating files,**

data consistency & multiple copy update module:

maintains data consistency and provides multiple copy update mechanisms,

transaction module: **provides transaction mechanism to allow grouping of elementary operations,**

recovery module: **provides recovery mechanism after failure,**

concurrency control module:

provides concurrency control mechanisms to ensure preservation of object consistency,

attribute access module: **reads and writes the file attributes,**

SERVICES OF DISTRIBUTED FILE SYSTEMS

DIRECTORY SERVICE

directory module		access control module
placement module	location module	relocation module

- **directory service maps user names into system names and protects files,**

- **directories usually organised into a tree to guarantee uniqueness of names and efficient name resolution,**

- **client requests access to a file using a directory service to locate the file that maps user names to addresses,**

- **directory service creates and modifies entries in directories, looks up user names in directories and returns the corresponding system names or file identifiers after checking the user's authorisation,**

- **directory is a file and a set of functions which distinguishes it from an ordinary data file,**

- **it has a fixed size and contains records that correspond to entries in the directory,**

- **file access is regulated by the authorisation service (or name service) which is located between the directory and file service,**

- **the directory service protects the files against unauthorised access using an access list for each file or capabilities,**

- **the authorisation service may be merged with either the file service or the directory service,**

- **responsibilities between file service and directory service are divided by using file identifiers,**

Directory and file naming

single global directory	**simple naming**
mounting directories	

1. single global directory

- **there is a single global directory where all the files belong to a single name space,**
- **this approach is unsuitable for DSs as a result of heterogeneous environment,**

2. simple naming

- **files are named by the server name, where files are located, and local names to guarantee the uniqueness of files,**
- **this approach is not location independent, i.e. moving a file from one computer to another requires changes in the filename and in the applications accessing that file,**

3. mounting directories

- **mounting remote directories to local directories provides transparent access only for the files placed in the mounted directory,**
- **once a remote directory is mounted, its files can be referenced in a location transparent manner,**

Access control of files

- access control checks file attributes to define whether the access to a file is permitted and supports updating of file attributes by using operations of the directory service,

- accessing and updating directories requires a special treatment of file attributes,

- directory service should provide operations which allow the owners of the files to grant and revoke access rights to themselves and to other users,

- file service uses a capability-based access control, i.e. clients must present an appropriate capability each time an operation on a file is requested,

- a file identifier contains a capability to access a file in order to provide selective access to reading, writing and the other file service operations,

- directory service requires clients to state their identity using a conventional identity-based approach, i.e. the directory service checks the users' authorisation whenever the client requests a file identifier using an access control list,

- the access control list is a table identifying the users and groups of users permitted to access the file and their permitted modes of access thus, each request to the directory service for a file identifier must include the user's identity and the modes of access requested,

Placement, location and relocation of files

- in a distributed system there can be more than one file server for increasing performance and reliability,

- a client should not need to be aware that there is more than one file server and files may be assigned to any of them,

- file servers must collaborate to provide an integrated file service, which is transparent to location, distribution and replication of files,

- there are two types of collaboration among file servers:

1. each file server is managing a partition of the total set of files and servers are located at separate locations,

2. replicas of files are distributed among several file servers

functions of the directory service:

1. placement of files:

- **a server must be chosen to hold the file when it is created,**

2. location of files:

- **the server which holds the given file must be found when the client wishes to refer to it,**

3. relocation of files:

- **this problem arises when files need to be relocated for performance or other reasons,**

Directory service operations

create_dir(*dir, parentdir*) - creates a new directory *dir* in the directory *parentdir*

delete_dir(*dir, parentdir*) - removes the empty directory *dir* from the directory *parentdir* and updates the attribute record

insert(*dir, name, file*) - adds the *(name, file)* pair to directory *dir*

un_name(*dir, name*) - removes an entry containing *name* from directory *dir* and decrements the reference count

open_dir(*dir*) - opens the directory *dir*

close_dir(*dir*) - closes the directory *dir*

read_dir(*dir, name, accessmode*) - returns the UFID of the file *name,* checking whether the client making the request is authorised to access that file in the manner specified in *accessmode*

get_names(*dir, pattern*) - returns the set of the names in directory *dir* that mach the regular expression given by *pattern*

rename(*dir, oldname, newname*) - the entry containing the *oldname* gets the *newname*

change(*olddir, oldname, newdir, newname*) - changes the entry from the old name to the new one

ch_dirattr(*dir, attributes*) - changes the access control attributes of directory *dir*

place(*file*) - finds the best file server for *file*

locate(*file*) - finds the address of the server holding *file*

REPLICATED FILE SERVICES

data consistency and multiple copy update module

- in a distributed system there can be more than one file server with several copies or replicas of files at different sites,

- all these servers must collaborate to provide an integrated file service, which is transparent to location, distribution and replication of files,

- a replicated file is a file which has a number of copies which are located in different servers,

- replication is expensive as a result of extra storage space required to store the replicas and the overhead incurred in maintaining all the replicas up to date,

- replication improves performance by reducing communication and reliability by increasing availability,

problems of file replication:

1. **replica is not updated due to a failure of the server storing the replica (consistency and multi-copy update problem),**

2. **all the file servers storing replicas are not reachable from all the clients due to network partition,**

tasks of file replication:

1. **keeping replicas of a file consistent,**

2. **detecting inconsistencies among replicas of a file and subsequently recovering from these inconsistencies,**

classes of collaborating file servers:

1. **each file server is managing a discrete partition of the total set of files (no replication)**

2. **file servers manage replicated files distributed along the DS,**

<u>Replication transparency</u>

- a repeated file service must function exactly like a non-replicated file service but must have better performance and reliability,

naming replicas

- **the replication transparency requires assigning the same identifier to all replicas of an object,**
- **this approach is suitable for immutable files, i.e. for files whose all copies are identical,**
- **however, different copies of mutable files may not be the same at a particular instance of time, as result the kernel cannot decide which replica is the most up-to-date one,**

replication control

- **it defines the number and locations of replicas of a related file,**

<u>explicit replication</u> users control the replication process in DFS,

i.e. when a file is created the user specify the server on which the file should be placed and on which additional copies can be created,

<u>implicit replication</u> the replication process is automatically controlled by the system without users' knowledge,

i.e. when a file is created, it does not provide any information about its location, the system selects one server for the file,

later the system automatically creates replicas of the as are necessary on other services,

File replication

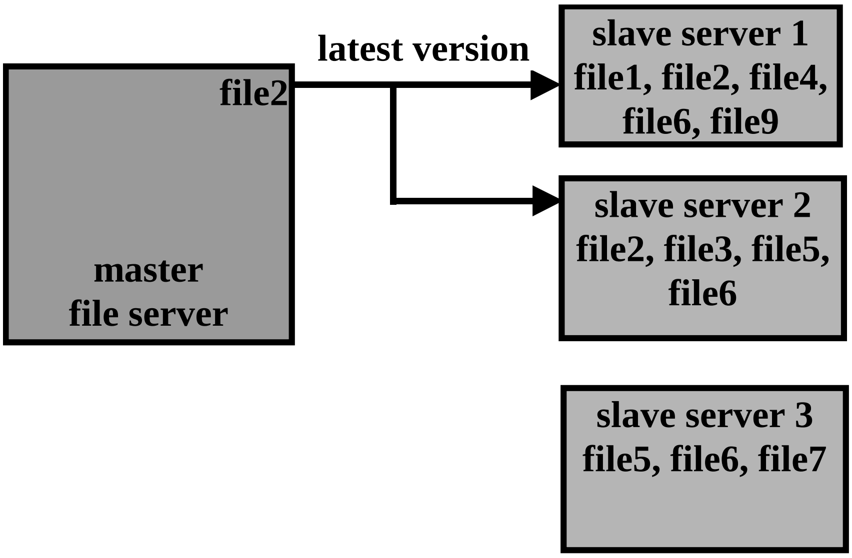
master/slave strategy	distributed multicopy update control

1. master/slave strategy

- **there is one master server, and one or more slave servers for each replicated file,**

- **the master server stores a master copy of the file and services all the update requests,**

- **copies of a file at slave servers is updated either by using copies of the master copy or receiving changes of the master file from the master server,**



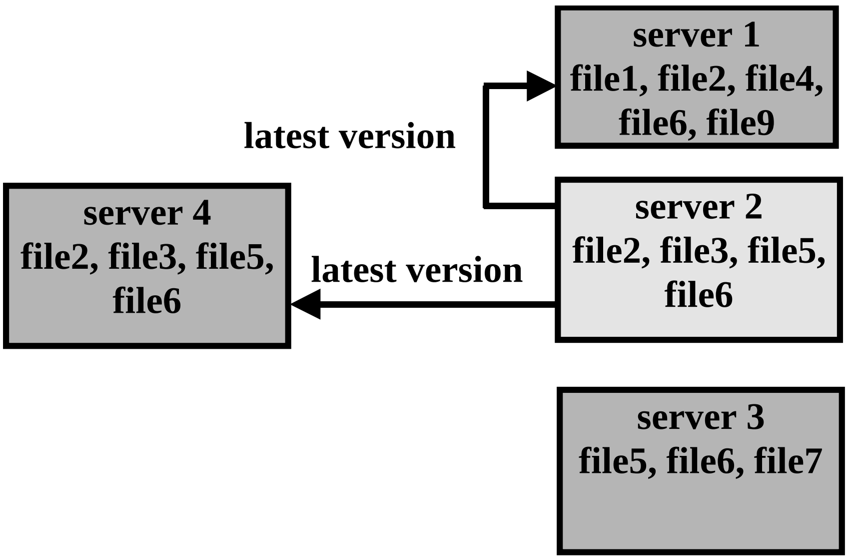
- **disadvantages of the strategy**

  - ♦ **updates must continue even when one or more servers containing copies of a file are unavailable,**

  - ♦ **all copies should be updated after changes in the master copy,**

2. distributed multi-copy update control

- **each server has the same rights in management of file replicas, i.e. there neither master nor slaves,**
- **each server, which maintains a copy of a file, accepts changes to that file and issues those changes to the other copies to ensure that clients have a consistent view of data,**
- **each server which holds a copy of a replicated file should maintain a list of copies, which contains file and server identifiers of the replicated files,**
- **servers must determine whether copies of a file are up-to-date before an operation is performed on the file,**
- **consistency of copies are defined by either timestamps or version numbers,**



- **multicopy update approaches:**

read-only replication	**read-any-write-all replication**
available copies replication	**primary copy replication**
quorom-based replication	

<u>read-only replication:</u>

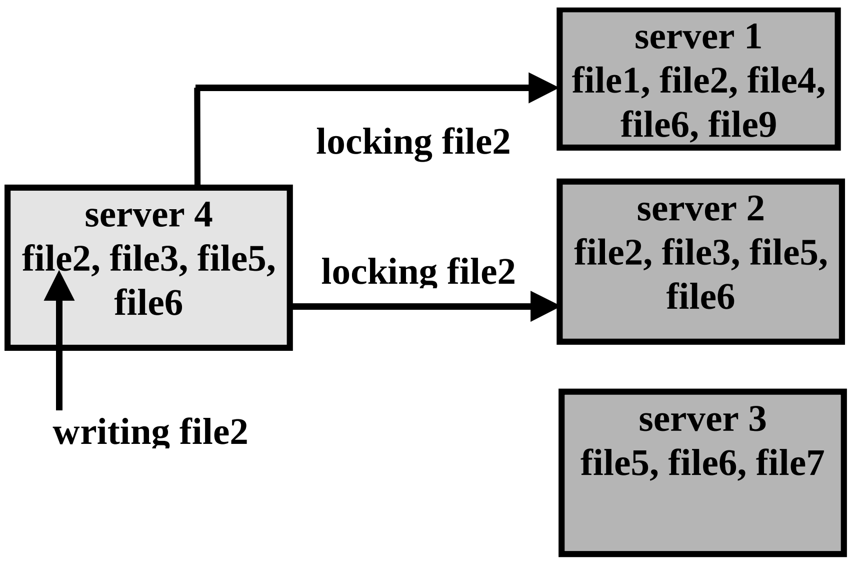
♦ **it supports replication of immutable files which are frequently read and non-frequently modified for example: system programs,**

<u>read-any-write-all replication:</u>

♦ **a read operation on a replicated file is performed by reading any copy of the file and a write operation by writing to all copies of the file,**

♦ **before updating any copy, all copies are locked, then they are updated and next the locks are unreleased,**

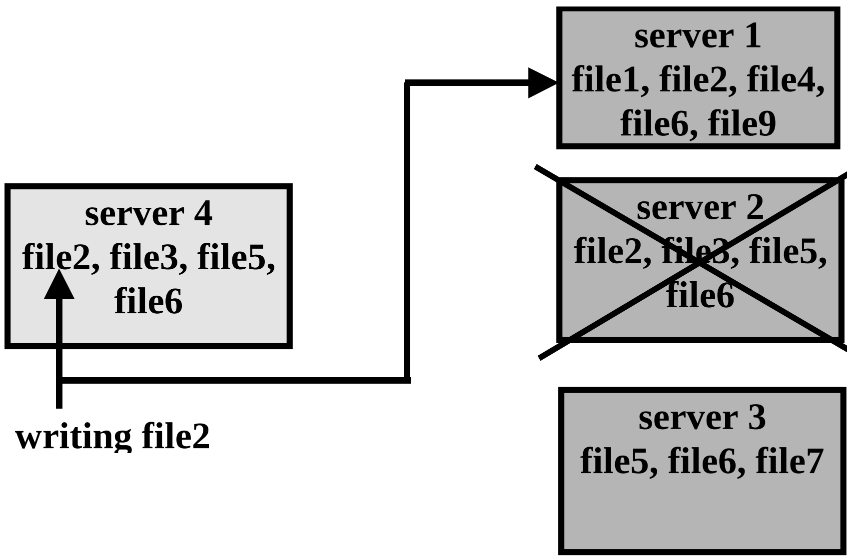
♦ **the write operation cannot be performed if any of the servers having a copy of the replicated file is down at the time of the write operation,**



<u>available copies replication:</u>

- **a read operation is performed be reading any available copies, but a write operations are executed by writing to all available copies excluding servers that are down,**

- **when a server recovers after a failure, it brings itself up-to-date by copying files from other servers before accepting any user read or write request,**



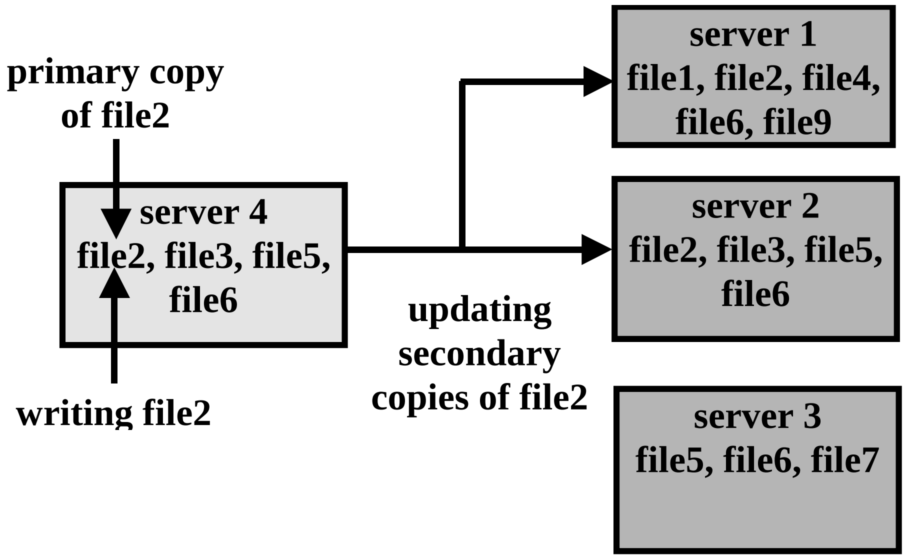
- **this replication approach does not prevent inconsistencies in the presence of communication failures,**

<u>primary copy replication:</u>

- **one copy is assigned to each replicated file as the primary copy all others are secondary copies,**

- **read operations can be performed using any copy, but all write operations are directly performed only on the primary copy,**

- **each server having the secondary copy updates its copy either by receiving notification of changes from the server having the primary copy or by requesting the updated copy from it,**



quorum-based replication:

- **if there are "n" copies of a replicated file, to read the file a minimum "r" copies of the file have to be consulted ("r" copies = set of read quorum),**

- **to perform write operation on the file a minimum "w" copies of the file have to be written ("w" copies = set of write quorum),**

- **to achieve consistency "r+w" should be greater than "n", i.e. there is at least one common copy of the file between any read/write quorum,**

- **up-to-date copies in a quorum are identified by version numbers, where the copy with the largest version number in a quorum is a current (or up-to-date),**

- **operations in this approach are not executed on all copies, therefore it becomes necessary to identify a current copy in a quorum using version number of copies,**

- **the version number is updated every time the copy is modified and the copy with the largest version number in a quorum is the current,**

read operation: **retrieve a read quorum (any "r" copies) of a file,**

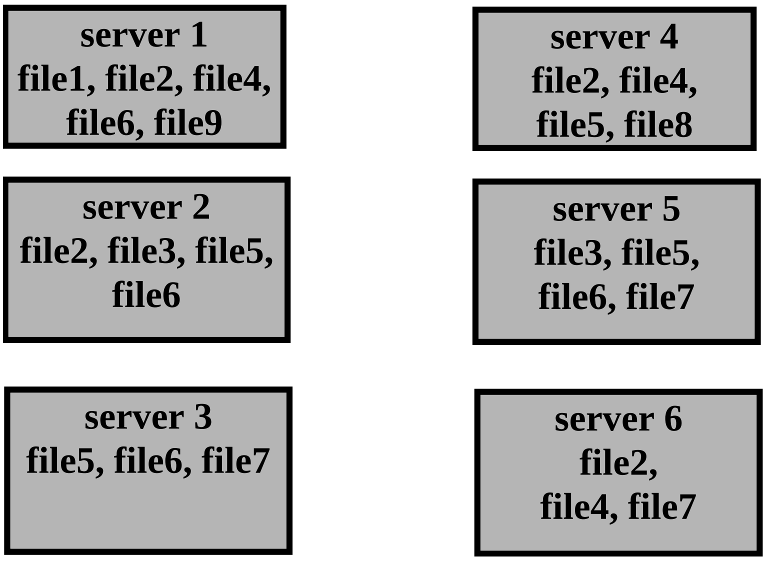
select the copy with the largest version number

perform the read operation on the selected copy,

write operation: **retrieve write quorum (any “w” copies) of a file,**

get the version number of the copy with the largest version number,

increment the version number and write the new value and the new version number to all “w” copies of the write quorum,



file No. 2 n = 4

write quorum w = 2 **writes files on server No.1 & No.6,**

these files receive the new values and version number,

read quorum r = 3 **has to select among files on server No. 1, No. 2 & No.4**

the read quorum has to include a copy on server either No.1 or No.6,

the read operation is performed using the copy of the file2 on server No.1,

TRANSACTION SERVICE

transaction module	recovery module	concurrency control module

- the transaction service is an extension of the file service that provides atomic actions on files,

- transaction is set of operations that take place indivisibly in concurrent computations and in the presence of failure,

- if a basic file service is used concurrently by a number of processes to read from or write data to the same file, unpredictable effects on the file can result,

- to avoid these problems, atomic transactions should be applied to the files, atomic transactions perform a high-level synchronisation to serialise access requests from multiple clients to maintain the shared file in consistent state,

- it means that basic file operations for example: create, open, write, close etc. should be grouped into a transaction using some primitives,

- the client process specifies a sequence of operations which are to be performed in one transaction and the transaction must preserve the atomicity of the entire sequence,

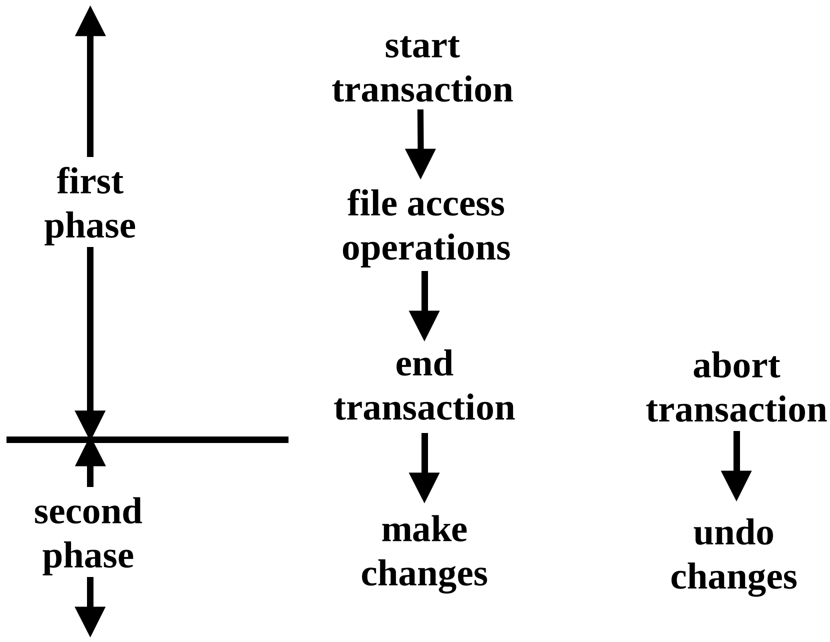
- each transaction must be recoverable, i.e. when the client or server stops the transaction must either be completed or all the updated files must be returned to their state before the transaction was started,

- if the transaction has progressed normally the client is informed by the server that the transaction is committed, otherwise, the transaction is aborted,

- if a server or a client process halts unexpectedly due to a hardware failure or software error before the transaction is completed, the server has to restore any files that were undergoing modification to their original states,

- the fact that a transaction can be aborted implies that all of the file operations must be permanently recorded or undone,

- a transaction has the following phases:



- in the first phase the file access operations in the transaction are performed and the client adds changes to file items,

- in the second phase the transaction is either committed or aborted,

- in the commit operation the changes made by the transaction to file items are made permanent, i.e. it will be visible to other transactions as well,

- in the abort operation the changes made by the transaction to file items are undone to restore the files to the state they were in before the transaction started,

Recovery methods

- the atomicity of transactions is threatened by the following failures:

- **transactions failures (erroneous data or processing),**
- **system failures (database, OS or hw failure),**
- **storage media failures,**

- there are two methods that reserve consistency and recover system after failures:

intention list approach	file version approach

1. intention list (or write-ahead log) approach

- **the server makes a list of all actions that change files while performing a transaction, i.e. the list is a log of operations of a transaction making changes to files,**

client operations	intention list
begin_transaction	
open(file)	
read(file)	
write(file)	write(file)
read(file)	
write(file)	write(file)
close(file)	
end_transaction	

- a record is created for each operation of the transaction, which modifies the file, the record is written to a log file, after this the operation is performed on the file to modify its contents,

- each record contains the transaction id, that is making the modification, the file id that is being modified, the items of the file that are being modified, and the old and new values of each item modified,

- if the transaction aborts, the information in the log file is used to roll back the individual file items to their initial values

2. file version approach

- **this approach ensures recoverability by avoiding overwriting the current file version in physical storage,**
- **the server makes new versions of the files containing the changes and if the transaction commits, the new version of the files is used, otherwise is discarded,**
- **when the first operation that modifies the file is performed within the transaction the server creates a tentative version of the file for the transaction from the current file version,**
- **the server performs all update operations on this file version during the transaction,**
- **a file has a sequence of versions which are the results of file modifications, each of these versions is written only once and is tentative,**
- **when the server commits a transaction, the most recently committed version of the file becomes the current version,**
- **if the transaction is aborted, the tentative file version is simply discarded and the current file version continues to remain the current version,**

transaction		versions	
begin_transaction			
open(file)	version 1		
read(file)			
write(file)		version 2	
read(file)			
write(file)			version 3
close(file)			
end_transaction			

- a transaction may modify more than one file and a file may be involved in several transactions, i.e. a file may have more than one tentative file version at the same time,

- when one of the concurrent transactions commits, the tentative version corresponding to that transaction becomes the current version of the file,

Concurrency control

- concurrency control must ensure that consistency of objects is preserved, and each atomic action is completed in finite time,

- consistency is provided by serialisation of atomic transactions, i.e. by ensuring that concurrently executing transactions do not interfere with each other,

- every transaction mechanism has to execute a concurrency control algorithm to prevent data inconsistency due to concurrent access by multiple transactions,

- approaches of the concurrency control:

locking	**optimistic concurrency control**
timestamps	

1. locking

- **server sets a lock, labelled with the transaction identifier, on each data item of a file before it is accessed,**
- **thus, only the transaction which locked the data item can access it, other transactions have to wait until the data item is unlocked,**
- **the data item is unlocked when the transaction is completed,**
- **locking is performed by the transaction service and clients have no access to operations for locking and unlocking data items,**

2. optimistic concurrency control

- **the server allows a transaction to proceed to the end of the first phase because access conflicts in concurrently executing transactions are very rare,**
- **before a transaction is completed, the server checks whether the transaction has used the same data items as an earlier transaction, in this case the transaction must be aborted and restarted,**

- **two records are kept to store the data items accessed within a transaction:**

  read set - data items read by the transaction,

  write set - data items created, changed or deleted by the transaction,

- **read and write set are compared with the write set of all of the concurrent transactions that reached the end of their first phase in order to validate a transaction,**

- **the validation fails if any data item present in the read or the write set of the transaction being validated is also present in the write set of the concurrent transactions,**

3. timestamps

- **execution of operations of the transaction, which follow the operation that caused the concurrency conflict, is actually waste of time,**

- **it is required to detect to operation that causes the concurrency conflict in order to abort the transaction at the point where the conflict occurs and it can be restarted,**

- **each transaction has a unique timestamp assigned to it after starting the transaction, every data item also has a read and a write timestamp,**

- **the server records the time when a data item was last read or wrote, i.e. when a transaction accesses a data item either its read or its write timestamp is updated to the transaction's timestamp,**

- **each transaction compares its own timestamp with that of the data item in order to define whether the current operation can be performed or not,**

- **before performing a read or a write operation on a data item, the server performs validation check by inspecting the timestamps on the data item,**

validation of read operation

- **if the timestamp of the transaction requested the read operation is more than the write timestamp of all committed and tentative values of the accessed data item, the read operation passes the validation check,**

- **the read operation can be performed only if there are no tentative values of the data items, otherwise it must wait until the completion of the transactions having tentative values of the data item,**

- **the validation check fails and the current transaction is aborted when either the timestamp of the current transaction is older than the timestamp of the most recent committed write to the data item or the timestamp of the current transaction is older than that of tentative value of the data item made another transaction, although it is more recent than the timestamp of the permanent data item,**

validation of write operation

– **if the timestamp of the transaction that requested the write operation is either equal to or more than the read and the committed write timestamps of the accessed data item, the write operation passes the validation check,**

- **if the timestamp of the current transaction is older than the timestamp of the last read or committed write of the data item, the validation fails, because another transaction has accessed the data item since the current transaction started,**

♦ **transactions are aborted and restarted when they are too late to do an operation,**

Transaction service operations

begin_transaction() - when the client wants a service, it sends a request,

the primitive checks whether the client has permission to start the transaction,

if the authorisation is OK then the server starts a new transaction and returns a unique transaction identifier *transid*,

all further messages within that transaction must refer the transaction identifier *transid*

end_transaction(*transid*) - after completing all the operations grouped into the transaction, the client sends the request to close (and commit) the transaction,

the returned result indicates whether the transaction has committed or failed, or whether the transaction is inactive because it is aborted,

abort_transaction(*transid*) - aborts the transaction,

the operation restores any change made within the transaction to the original values, and the state of the transaction is set to inactive,

tread(*transid, fname, position, n buffer*) – returns to the client *n* bytes of the tentative data resulting from the *transid* any if has been recorded,

twrite(*transid, fname, position, n buffer*) – performs the same operation as write but records the new data in a tentative form that is made permanent when only when the *transid* commits,

FILE SERVICE

file module	file access module	attribute access module

- file is a sequence of data items with a given size, which may be accessed and modified by read and write operations,

- files also have attributes that describe properties of a file (e.g. file name, allowed operations, access list, date and time of creation, last modification and last read, owner, encryption key)

- file service provides an abstraction of files, each of which is a sequence of data items (records),

- basic file operations provided by the file service are the read/write of data blocks and get/set of file attributes,

- file service also supports creation and deletion of files by adding/deleting in the directory and allocating/de-allocating buffer and file spaces,

- the user is not concerned with how and/or where the data in the file are stored or with any relationships between files,

Access to files

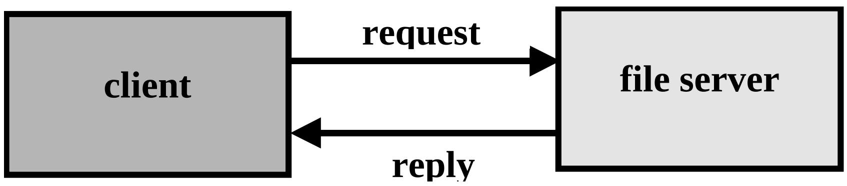
- access to a file depends on the file accessing models used by the file system that are defined by two factors:

accessing remote files	**data transfer unit**

accessing remote files:

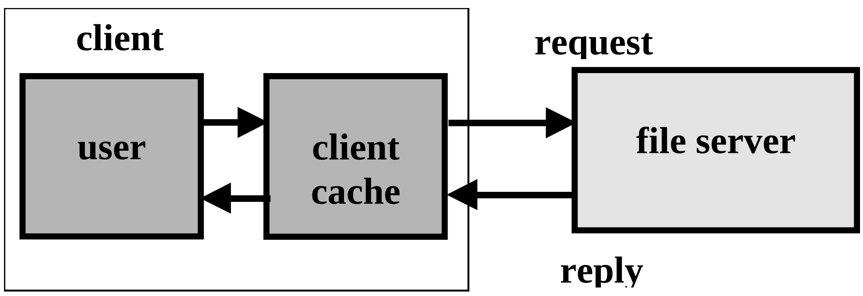
1. remote service model

- **the client's request for file access is delivered to the server, the server performs the access request and forwards back the result to the client,**
- **the access requests and server replies are transferred across the network that may impose significant communication overheads**



2. data caching model

- **if the data is needed for a client is not accessible locally, it is copied from the server node to the client node's cache,**
- **the client's obtains the data from the cache decreasing the communication overload because recently accessed data are stored in the cache for some time to provide local access to it,**



- **data of any file may simultaneously be available in different caches, thus write operations, which modify the data, require that changes must be made in both the original file at the server node and any other caches storing the data**

- **many DFSs use data caching approach to increase performance and to achieve greater scalability, or hybrid method combining remote service and data caching,**

data transfer units:

file-level transfer model	**block-level transfer model**
record-level transfer model	**byte-level transfer model**

1. file-level transfer model

- **the whole file, containing the requested data, is transferred across the network between the client and the server,**

- **transfer of the complete file requires only one request resulting in reduced server load and network traffic,**

2. block-level transfer model

- **blocks are sent as a part of the data transfer across the network between the client and the server,**

- **this model does not require large storage capacity at the client node because only the requested part of the file is transferred,**

- **when an entire file is to be accessed, multiple server requests are needed that increase communication overhead,**

3. record-level transfer model

- **it is applied to those files which file contents are stored in records in, i.e. it is a structure-based model**

4. byte-level transfer model

- **bytes are moved between the client node and the server node across the network,**

- **this model provides maximum flexibility because it allows storage and retrieval of an arbitrary sequential sub-range of a file,**

File sharing

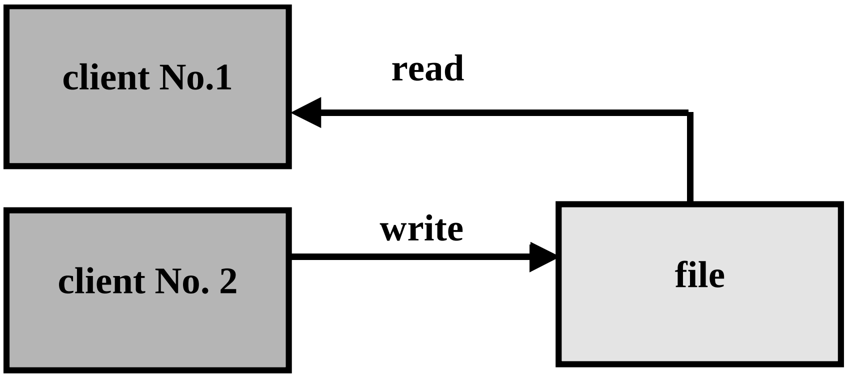
- a shared file may be simultaneously accessed by multiple users that requires informing all users about modifications of data in the file,

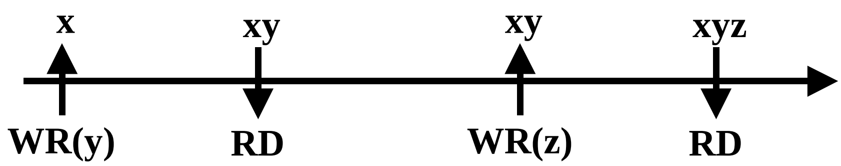
- there are several file sharing methods that provide this property:

UNIX file sharing method	**session method**
immutable shared file method	**transaction-based method**

1. UNIX file sharing method

- **it enforces an absolute time ordering on all operations and ensures that every RD operation on a file see effects of all previous write operations performed on that file,**





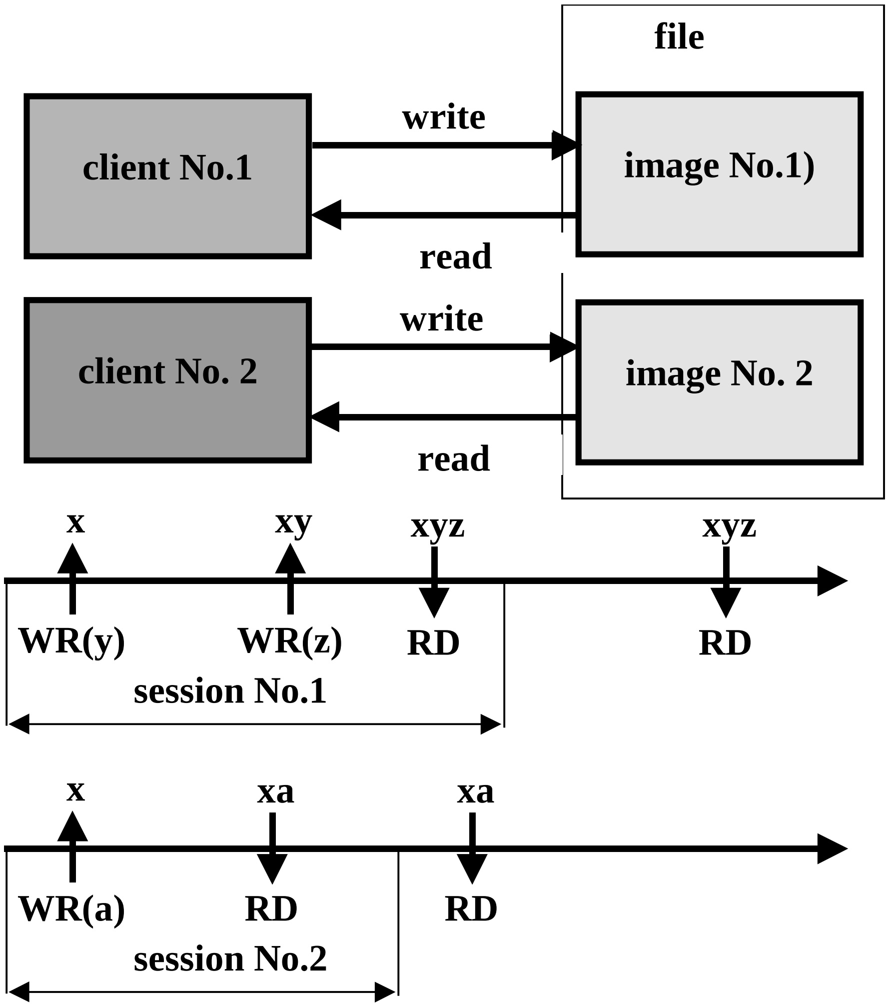
- **there several problems while implementing the UNIX file sharing method to DSs:**

  a. client requests from different nodes may arrive and process at the sever node in an order different from the actual order in which the requests were made due to network delays,

  b. a file may be managed by more than one file server and it may be stored by several caches at client nodes,

2. session method

- **a client performs a session, i.e. he/she opens a file, performs RD/WR operations on the file, and closes the file,**



- **all changes made to the file during the session are visible only to the client process that opened the session, and are invisible to other remote processes who have opened the same file simultaneously, i.e. each client maintains its own image of the file,**

- **there is an open question what will be the final image of the file when multiple users sessions are closed one after another?**

- **session method is suitable for those file systems that use the file-level transfer model,**

3. immutable shared file method

- **as it is based on the immutable file model, it allows files to be shared only in the read-only mode,**

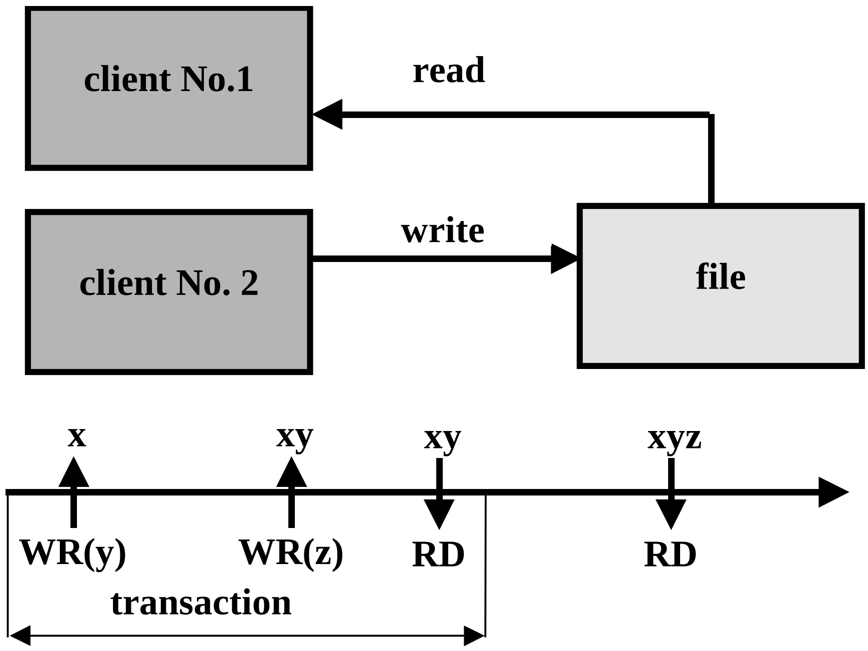
- **once a file is declared as shareable to file is managed as immutable, i.e. it cannot be modified anymore,**

- **as a result, the problem of file sharing disappears,**

4. transaction-based method

- **the transactions ensure that the partial modifications made to the shared data by a transaction will not be visible to other concurrently executing transactions until the transactions ends,**

- **thus, the final content of the file will be the same as if all transactions were run in some sequential order,**



Fault tolerance

- there could be several faults in DSs that affect data integrity, for example

- **contents of the main memory may be lost as a result of a crash,**
- **server or client node may crash losing the state information of files open for access,**
- **transient fault or decay of disk storage may result in loss or corruption of data stored in the file system,**

- storage approaches:

1. volatile storage

- **data stored in a volatile storage, for example in RAM, is lost after a power failure or machine crash,**

2. non-volatile storage

- **disks provide non-volatile storage that withstands CPU failures but cannot withstand transient I/O faults and decay of the storage media,**

3. stable storage

- **it uses duplicate storage devices to provide stable device in order to withstand transient I/O faults and decay of the storage media,**
- **a disk-based storage system consists of two disks where each block on a disk is an exact copy of the corresponding block on other disk,**
- **a read operation first attempts to read from one of the disk, if it fails it is performed on the other disk,**
- **a write operation writes both disks, but the write to the second disk does not start until that for the first disk has been successfully completed,**
- **after a crash a recovery action compares the contents of the two**

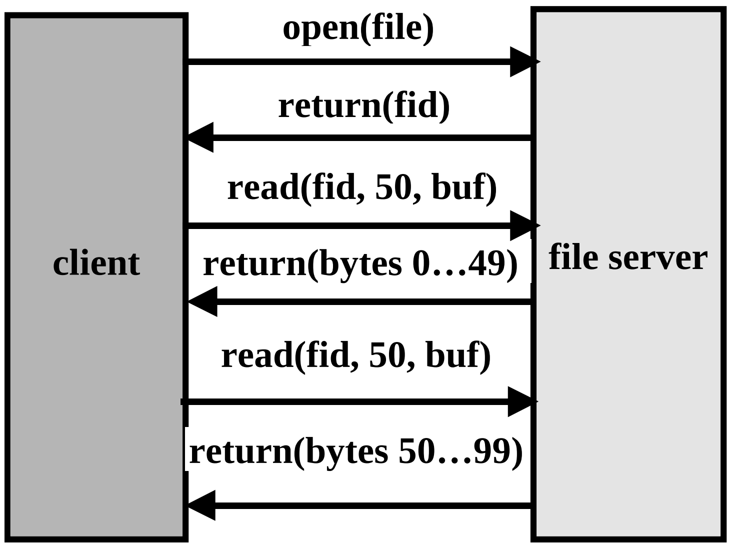
disks block by block, whenever blocks differ, the blcok having incorrect data is regenerated from the corresponding block on other disk,

service paradigm and fault tolerance:

stateful servers	stateless servers

1. stateful file servers

- **it maintains clients' state information from one access request to the next,**



- **the server creates state information for a client when the client starts a new session by performing an open operation, and discards the state information when the client closes the session by performing a close operation,**

2. stateless file servers

- **stateless server is a server which doesn't maintain its internal state. As a result, the parameters, which are necessary to perform an operation, are stored by the client, and is passed to the server in each request message,**

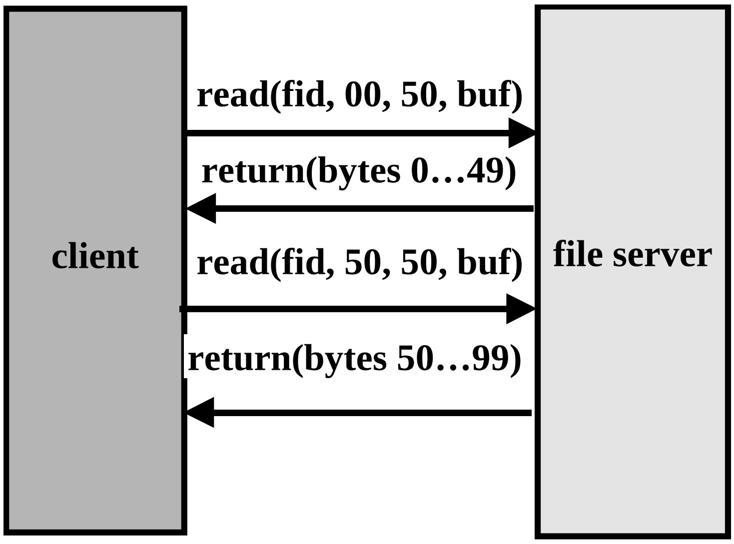
- **thus, each request identifies the file and the position in the file for the read/write operation,**

- **the server does not keep track of any past request. After opening a file, the index to this entry is returned to the client for use in**

subsequent request, for example: read-write pointer,

- **a stateless server also supports crash recovery, i.e. when a server crashes, the client retransmits only its request message until a response message is received. When the client crashes, no recovery is necessary.**

- **recovery using a server with state information is much harder because the client must detect server crashes in order to rebuild the server state, and the server must detect client crashes in order to discard any state it is holding for the client,**



- **it is difficult to implement a normal file system in a fully stateless way (e.g. deleting an open file, change permissions on file),**

- **using stateless file servers it is easy to perform crash recovery because no client state information is maintained by the server and each request contains all information that is necessary to complete the request,**

- **when a server crashes while serving a request, the client need only re-send the request until the server responds, and the server does no crash recovery at all,**

- **when a client crashes during request processing, no recovery is necessary for either the client or the server,**

File service operations

- a file contains data and has attributes as a result there are two separate sets of operations, one to deal with the data and the other with attributes.

file services operations on data:

create(*file)* **creates a new *file* of zero length with a specified name, establishes its default attributes, and assigns a UFID to it,**

delete(*file*) **removes the *file* from the file store.**

open(*file*) **opens the existing *file,* that can be used to perform different operations on it,**

close(*file*) **terminates access for actions to be taken on *file***

read(*file, offset, count, buffer*): reads up to *count* bytes of data from *file* starting *offset* bytes and puts them continuously into *buffer*

write(*file, offset, count, buffer*): writes *count* bytes of data from *buffer* to *file* beginning *offset* bytes from the beginning of the *file*

copy(*source, destination*): copies *source file* to *destination file*

get_size(file) **return the file size**

file service operations on attributes

read_attribute(*file*): returns the values of the attributes of the *file*

write_attributes(*file, attribute*): modifies the existing value of the *attribute* of the *file*

DISK SERVICE

block module	device module	caching module

- is the interface to the hardware of the file system that provides a logical view of the file system,

- enables a single storage system to be shared between different file services,

- supports reading or writing raw data blocks without regard to how they are organised,

- provides the following functions for users:

- **mapping of logical address to physical block addresses,**
- **interfacing to services at the device level for data block allocation/de-allocation,**
- **read/write operations of data blocks of the file system,**

- in many DFSs the disk service is separated from the file service in order to easy combine different methods of storage, and different storage media because different file servers can coexist and share the same disk storage device in a single distributed system,

- a file can be stored in a non-contiguous set of blocks of storage (this implies the need for recording a sequence of pointers (file index) to the blocks or other method,

- a translation is necessary from user name to UFID, and from UFID to block pointer,

- file service translates UFIDs to file server location and file addresses thus, users don’t need to know the location of the files,

- to ensure location and migration transparency UFIDs must not include file server locations or file addresses,

File storage

- **files are stored in non-contiguous sets of blocks with fixed size by the storage device,**

- **file system contains a file index which consists of sequences of pointers to the blocks of a file**

- **file index contains the length and attributes of files where the attribute record is managed by the directory service**

- **every file maintained by a file system has at least one user name and a unique system name,**

- **several servers may provide a file service thus, file identifiers – UFIDs - must be unique in a distributed system,**

- **UFIDs are not addresses thus, they do not include the network address of the server that holds the file,**

- **UFIDs should be unique and difficult to forge thus, they should use large and sparsely populated number space,**

- **UFID structure:**

server (or group) identifier	**file number**	**random number**

server identifier	**designates the file group,**
file number	**file identifier which is incremented each time a file is created in a file group and cannot be reused to provide uniqueness of UFIDs,**
random number	**to check the validity of requests to access the file** **the number is not accessible to clients**

File caching

- the performance of any computer system partially rely on the number of disk access operations and remote operations, it is necessary to avoid them whenever it is possible,

- file cache retains recently accessed file data in main memory, so that repeated accesses to the same information can be handled without additional disk transfers,

- problems of caching systems in centralised file systems:

- **granularity of cached data,**
- **cache size,**
- **replacement policy,**

- problems of caching systems in distributed file systems:

- **cache location,**
- **cache modification propagation,**
- **cache validation,**
- **reliability after system crashes,**
- **overload avoidance and management,**
- **performance and consistency issues (how to avoid duplicated caching),**

cache consistency and performance:

passive	**server maintains cached objects**
server monitors object access	

1. passive

- **do nothing, it is the responsibility of the users to be careful,**

2. server maintains cached objects

- **the server keeps track of which client has which blocks and files in its cache,**
- **when a client modifies a block or a file, it informs the file server,**
- **the file server informs the other clients that the particular block or file is invalid and has to be removed from their cache,**

3. server monitors object access

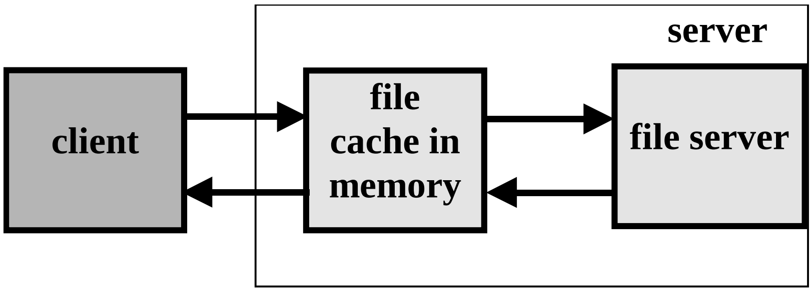
- **clients can access only files in the cache,**
- **a client informs the file server when opening a file indicating whether this is to be opened for reading or writing,**
- **server must react only if one of the processes writes the particular file and there is another process reading that file at the same time. In this case the server asks the writing process to upload the modified file that can be accessed by the others.**
- **only efficient if the shared files are rarely updated (in most cases they are),**

cache location:

server's main memory	**client disk**
client's main memory	

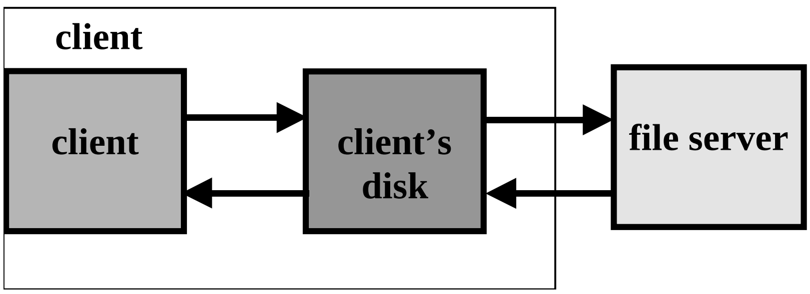
<u>1. cache in server's main memory</u>

- **a cache in the server's main memory eliminates the repeated disk access at the server node,**
- **this approach keeps the original file and cached data consistent since both reside on the same node,**
- **single server manages both the cached data and the file, thus multiple accesses from different clients can be easily synchronised,**
- **it requires a network operation for each file access by a remote client and processing of the access request by the server that does not supports reliability and scalability,**



<u>2. cache in client disk</u>

- **this approach eliminates network access but requires disk access at each file request,**



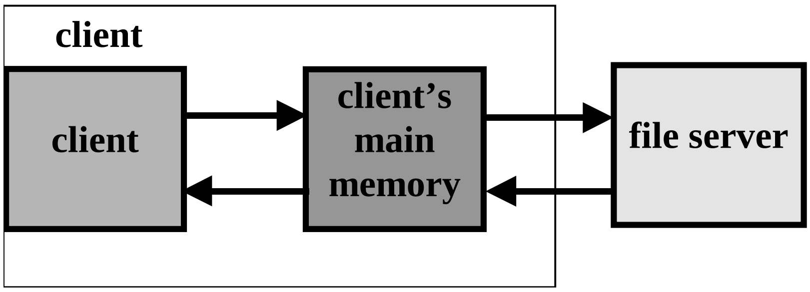
- **it requires large storage capacity, thus it support file-level data transfer model in DSs,**

- ♦ **it ensures disconnected operation and contributes to reliability and scalability,**

3. cache in client's main memory

- ♦ **it eliminates both disk access and network access, thus it provides maximum performance comparing with the first two approaches,**

- ♦ **it also permits the use of diskless workstations in distributed systems**



cache modification propagation:

- **in DSs, which cache is located on client node, data in a file may be simultaneously be cached on multiple nodes in,**

- **keeping file data cached at multiple client nodes consistent is an important issue in DSs,**

- **there are several questions that should be answered in order to provide data consistency:**

  - ♦ **when to propagate modifications made to a data in a cache to the corresponding file server,**

  - ♦ **how to verify the validity of a data in a cache,**

data modification propagation methods:

write-through scheme	**delayed-write scheme**

1. 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,**

- **each write access has to wait until the data is written to the master copy of the server,**

- **this scheme provides advantages of data caching only for read accesses because it does not reduce write accesses through the network to the file server,**

2. delayed-write scheme

- **to decrease communication via network, when a data in a cache is modified, the new value is written only to the cache and the client just makes a note that the cache entry has been updated,**

- **at later phase, all updated data of files in the cache are gathered together and sent to the server at a time,**

- **there are three delayed-write methods:**

  a. write on ejection from cache

  - the cache replacement policy decides when to move the cache entry to the server,

  b. ***periodic write*** **- the cache is scanned regularly and any cached data that have been modified since the last scan are sent to the server,**

  c. ***write one close*** **– the modifications of the cached data are sent to the server when the file is closed by the client,**

cache validation:

- **the modification policy only specifies when the master copy of a file at the server node is updated, it does not manage updating of copies of the file in cache of other nodes,**

- **thus, it is necessary to verify if data (or file) in a cache at a client node is consistent with the master copy,**

- **cache validation methods:**

client-initiated approach	**server-initiated approach**

1. client-initiated approach

- **a client contacts the server and checks whether is locally cached data is consistent with the master copy,**

- **there are three client-initiated methods:**

  a. ***checking before every access***

  - a check is initiated before every access

  b. ***periodic checking*** **- a check is initiated at fixed interval of time,**

  c. check on file open

  - a client's cache is validated when the client opens the file

- **the client-initiated cache validation requires intensive network communication and consumes CPU time if the validity check is performed frequently,**

2. server-initiated approach

- **a client informs the file server when opening a file indicating whether the file is 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 keeps monitoring the file modes being used by different clients and reacts whenever it detects a potential for inconsistency,**

Disk service operations

- disk service basically provides allocating new blocks, reading or writing and releasing blocks.

- disk services operations:

allocate_block() - allocates a new block and delivers its pointer

free_block(*block*) - releases a *block* that is no longer required

get_block(*block*) - the specified *block* is read from disk and stored in a free buffer of the cache, and a copy is sent to the calling process

if the *block* is already in the cache, no access is made.

put_block(*block, data*) - *data* stored in the cache is copied to disk at the given address *block*

flush_block(*block*) - removes *block* from the cache.

- during read and write operations checksums are used to detect errors.