DISTRIBUTED FILE SYSTEM

OBJECTIVE: To explain the approaches for designing a file system

OUTCOME: Able to discuss different types of file system and approaches used for designing them

Two main purposes of using files:

1. Permanent storage of information on a secondary storage media.
2. Sharing of information between applications.

A file system is a subsystem of the operating system that performs file management activities such as organization, storing, retrieval, naming, sharing, and protection of files.

A **Distributed File System** (DFS) is simply a classical model of a file system distributed across multiple machines. The purpose is to promote sharing of dispersed files.

The distributed file system supports the following:

1. Remote information sharing

Thus any node, irrespective of the physical location of the file, can access the file.

2. User mobility

User should be permitted to work on different nodes.

3. Availability

For better fault-tolerance, files should be available for use even in the event of temporary failure of one or more nodes of the system. Thus the system should maintain multiple copies of the files, the existence of which should be transparent to the user.

4. Diskless workstations

A distributed file system, with its transparent remote-file accessing capability, allows the use of diskless workstations in a system.

A distributed file system provides the following types of services:

1. Storage service

Allocation and management of space on a secondary storage device thus providing a logical view of the storage system.

2. True file service

Includes file-sharing semantics, file-caching mechanism, file replication mechanism, concurrency control, multiple copy update protocol etc.

3.Name/Directory service

Responsible for directory related activities such as creation and deletion of directories, adding a new file to a directory, deleting a file from a directory, changing the name of a file, moving a file from one directory to another etc.

Desirable features of a distributed file system:

1. Transparency:

i) Structure transparency

Clients should not know the number or locations of file servers and the storage devices. Note: multiple file servers provided for performance, scalability, and reliability.

ii) Access transparency

Both local and remote files should be accessible in the same way. The file system should automatically locate an accessed file and transport it to the clients site.

iii) Naming transparency

The name of the file should give no hint as to the location of the file. The name of the file must not be changed when moving from one node to another.

iv) Replication transparency

If a file is replicated on multiple nodes, both the existence of multiple copies and their locations should be hidden from the clients.

Continued..

2. User mobility:

Automatically bring the users environment (e.g. users home directory) to the node where the user logs in.

3. Performance:

Performance is measured as the average amount of time needed to satisfy client requests.

This time includes CPU time + time for accessing secondary storage + network access time.

It is desirable that the performance of a distributed file system be comparable to that of a centralized file system.

Continued..

4. Simplicity and ease of use

User interface to the file system be simple and number of commands should be as small as possible.

5. Scalability

Growth of nodes and users should not seriously disrupt service.

6. High availability

A distributed file system should continue to function in the face of partial failures such as a link failure, a node failure, or a storage device crash.

A highly reliable and scalable distributed file system should have multiple and independent file servers controlling multiple and independent storage devices.

Continued..

7. High reliability

Probability of loss of stored data should be minimized. System should automatically generate backup copies of critical files.

8. Data integrity

Concurrent access requests from multiple users who are competing to access the file must be properly synchronized by the use of some form of concurrency control mechanism. Atomic transactions can also be provided.

9. Security

Users should be confident of the privacy of their data.

10. Heterogeneity

There should be easy access to shared data on diverse platforms (e.g. Unix workstation, Wintel platform etc).

File Model

1. Unstructured and Structured files:

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

Continued..

2. Mutable and immutable files:

- Based on the modifiability criteria, files are of two types, mutable and immutable.
- An **update performed on a file overwrites its old contents to produce the new contents**.(ex. Unix and MSDOS)

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

File Accessing Models

Depends on the method used for accessing remote files and the unit of data access.

1. Accessing remote files

A distributed file system may use one of the following models to service a clients file access request when the accessed file is remote:

a. Remote service model

- Processing of a clients request is performed at the **servers node.**
- The clients request for file access is delivered **across the network as a message to the server**, the server machine performs the access request, and the result is sent to the client.
- Need to **minimize the number of messages sent** and the overhead per message.

File Accessing Models

b. Data-caching model

- This model attempts to **reduce the network traffic** of the previous model **by caching the data obtained from the server node.**
- This takes advantage of the **locality feature** of the found in file accesses.
- A replacement policy such as **LRU is used to keep the cache size bounded**.
- While this model **reduces network traffic** it has to deal with the **cache coherency problem during *writes*,**
- Because the **local cached copy of the data needs to be updated**, the original file at the server node needs to be updated and copies in any other caches need to be updated.

Continued..

Advantage of Data-caching model over the Remote service model:

The data-caching model offers ,

- Possibility of **increased performance**
- Greater **system scalability** because it reduces network traffic,
- contention for the network, and contention for the file servers.

Hence almost all distributed file systems implement some form of caching.

Example: NFS uses the remote service model but adds caching for better performance.

Continued..

Unit of Data Transfer:

- In file systems that use the data-caching model, an important design **issue is to decide the unit of data transfer.**
- This refers to the fraction of a file that is transferred to and form clients as a result of single read or write operation.

i) File-level transfer model

- In this model when file data is to be transferred, the entire file is moved.

Advantages:

- More efficient than transferring page by page which requires more network protocol overhead.
- Reduces server load and network traffic since it accesses the server only once.
- This has better scalability.
- Once the entire file is cached at the client site, it is immune to server and network failures.

Disadvantage:

- Requires sufficient storage space on the client machine.
- This approach fails for very large files, especially when the client runs on a diskless workstation.
- If only a small fraction of a file is needed, moving the entire file is wasteful.

Continued..

ii) Block-level transfer model:

- File transfer takes place in file blocks.
- A file block is a contiguous portion of a file and is of fixed length (can also be a equal to a virtual memory page size).

Advantages:

- Does not require client nodes to have large storage space.
- It eliminates the need to copy an entire file when only a small portion of the data is needed.

Disadvantages:

- When an entire file is to be accessed, multiple server requests are needed, resulting in more network traffic and more network protocol overhead.
- NFS uses block-level transfer model.

Continued..

iii) Byte-level transfer model:

- Unit of transfer is a byte.
- Model provides maximum flexibility because it allows storage and retrieval of an arbitrary amount of a file, specified by an offset within a file and length.

Disadvantages:

- Cache management is harder due to the variable-length data for different access requests.

iv)Record-level transfer model:

- This model is used with structured files and the unit of transfer is the record.

File-Accessing Models

- Accessing Remote Files

	File access	Merits	Demerits
Remote service model	At a server	A simple implementation	Communication overhead
Data caching model	At a client that cached a file copy	Reducing network traffic	Cache consistency problem

- Unit of Data Transfer

Transfer level	Merits	Demerits
File	Simple, less communication overhead, and immune to server	A client required to have large storage space
Block	A client not required to have large storage space	More network traffic/overhead
Byte	Flexibility maximized	Difficult cache management to handle the variable-length data
Record	Handling structured and indexed files	More network traffic More overhead to re-construct a file.

File Caching Schemes

Every distributed file system uses some form of caching. The reasons are:

1. Better performance since repeated accesses to the same information is handled additional network accesses and disk transfers. This is due to locality in file access patterns.

2. It contributes to the scalability and reliability of the distributed file system since data can be remotely cached on the client node.

File-caching scheme for distributed systems:

1. Cache location
2. Modification Propagation
3. Cache Validation

Cache Location

- This refers to the place where the cached data is stored.
- Assuming that the original location of a file is on its servers disk, there are three possible cache locations :

1. Servers main memory
   - In this case a cache hit costs one network access.
   - It does not contribute to scalability and reliability of the distributed file system.
   - Since we every cache hit requires accessing the server.

Advantages:

a. Easy to implement
b. Totally transparent to clients
c. Easy to keep the original file and the cached data consistent.

Cache Location

2. Clients disk

- In this case a cache hit costs one disk access.
- This is somewhat slower than having the cache in servers main memory.
- Having the cache in servers main memory is also simpler.

Advantages:

a. Provides reliability against crashes since modification to cached data is lost in a crash the cache is kept in main memory.

b. Large storage capacity.

c. Contributes to scalability and reliability because on a cache hit the access request can be serviced locally without the need to contact the server.

Cache Location

3. Clients main memory

- Eliminates both network access cost and disk access cost.
- This technique is not preferred to a clients disk cache when large cache size.
- Increased reliability of cached data are desired.

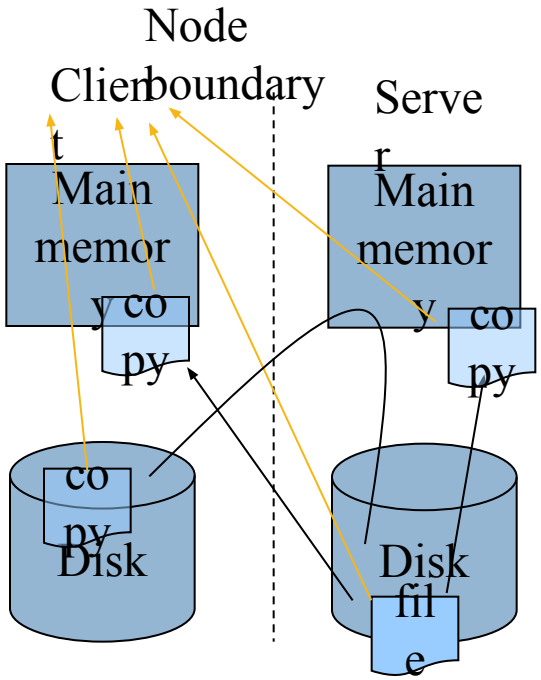
Advantages:

a. Maximum performance gain.

b. Permits workstations to be diskless.

c. Contributes to reliability and scalability.

Summary of File Caching



Location	Merits	Demerits
No caching	No modifications	Frequent disk access, Busy network traffic
In server's main memory	One-time disk access, Easy implementation, Unix-like file-sharing semantics	Busy network traffic
In client's disk	One-time network access, No size restriction	Cache consistency problem, File access semantics, Frequent disk access, No Diskless workstation
In client's main memory	Maximum performance, Diskless workstation, Scalability	Size restriction, Cache consistency problem, File access semantics

Modification Propagation

- When the cache is located on clients nodes, a files data may simultaneously be cached on multiple nodes.

- It is possible for caches to become *inconsistent* when the file data is changed by one of the clients and the corresponding data cached at other nodes are not changed or discarded.

There are two design issues involved:

1. When to propagate modifications made to a cached data to the corresponding file server.
2. How to verify the validity of cached data.

Modification Propagation

Techniques used include:

i)Write-through scheme:

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

Advantage:

- High degree of reliability and suitability for UNIX-like semantics.
- Data getting lost in the event of a client crash is very low since every modification is immediately propagated to the server having the master copy.

Disadvantage:

- This scheme is only suitable where the ratio of read-to-write accesses is fairly large.
- It does not reduce network traffic for writes.
- This is due to the fact that every write access has to wait until the data is written to the master copy of the server.
- Hence the advantages of data caching are only read accesses because the server is involved for all write accesses.

Modification Propagation

ii) Delayed-write scheme.

- To reduce network traffic for writes the delayed-write scheme is used.

- In this case, the new data value is only written to the cache and all updated cache entries are sent to the server at a later time.

There are three commonly used delayed-write approaches:

i. Write on ejection from cache

ii. Periodic write

iii. Write on close

Delayed write scheme

i. Write on ejection from cache

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

ii. Periodic write

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

iii. Write on close

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

Delayed write scheme

Advantages of delayed-write scheme:

- 1. Write accesses complete more quickly because the new value is written only client cache. This results in a performance gain.
- 2. Modified data may be deleted before it is time to send them to the server (e.g. temporary data),results in a major performance gain.
- 3. Gathering of all file updates and sending them together to the server is more efficient than sending each update separately.

Disadvantage of delayed-write scheme:

- Reliability can be a problem since modifications not yet sent to the server from a clients cache will be lost if the client crashes.

Cache Validation schemes

- The modification propagation policy only specifies when the master copy of a file on the server node is updated upon modification of a cache entry.
- It does not tell anything about when the file data residing in the cache of other nodes is updated.
- A clients cache entry becomes stale as soon as some other client modifies the data corresponding to the cache entry in the master copy of the file on the server.
- It becomes necessary to verify if the data cached at a client node is consistent with the master copy.
- If not, the cached data must be invalidated and updated data must be fetched again from the server.
- There are two approaches to verify the validity of cached data:
  - Client-initiated approach.
  - Server-initiated approach.

Client-initiated approach

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

- Two approaches may be used:

1. Checking before every access.

- The server needs to be contacted on every access.

2. Periodic checking.

- A check is initiated every fixed interval of time.

Disadvantage :

If frequency of the validity check is high,

- Generates a large amount of network traffic
- Consumes precious server CPU cycles.

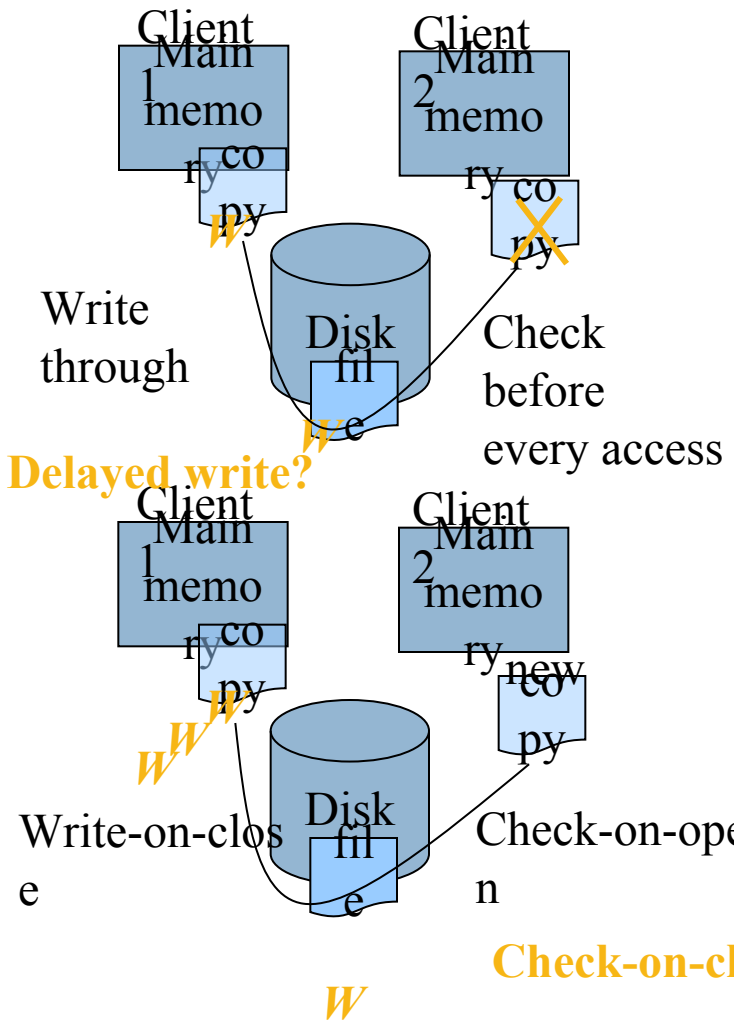
Server-initiated approach

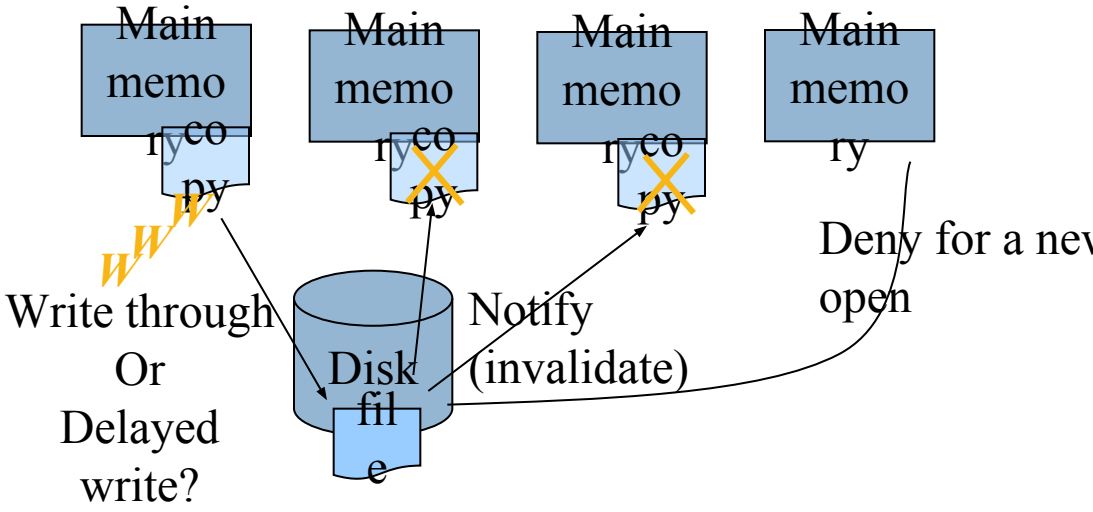
- A client informs the file server when opening a file, indicating whether a file is being opened for reading, writing, or both.
- The file server keeps a record of which client has which file open and in what mode and reacts whenever it detects a potential for inconsistency
- E.g. if a file is open for reading, other clients may be allowed to open it for reading, but opening it for writing cannot be allowed.
- When a client closes a file, it sends intimation to the server along with any modifications made to the file. Then the server updates its record of which client has which file open in which mode.
-
- When a new client makes a request to open an already open file and if the server finds that the new open mode conflicts with the already open mode, the server can deny the request, queue the request, or disable caching by asking all clients having the file open to remove that file from their caches.

Disadvantage:

- It requires that file servers be stateful.

Client-initiated approach /Server Initiated approach





File Replication

- A *replicated file* is a file that has multiple copies, with each file on a separate file server.
- Difference Between Replication and Caching
  - A replica is associated with a server, whereas a cache with client.
  - A replicate focuses on availability, while a cache on locality
  - A replicate is more persistent than a cache is
  - A cache is contingent upon a replica

File Replication

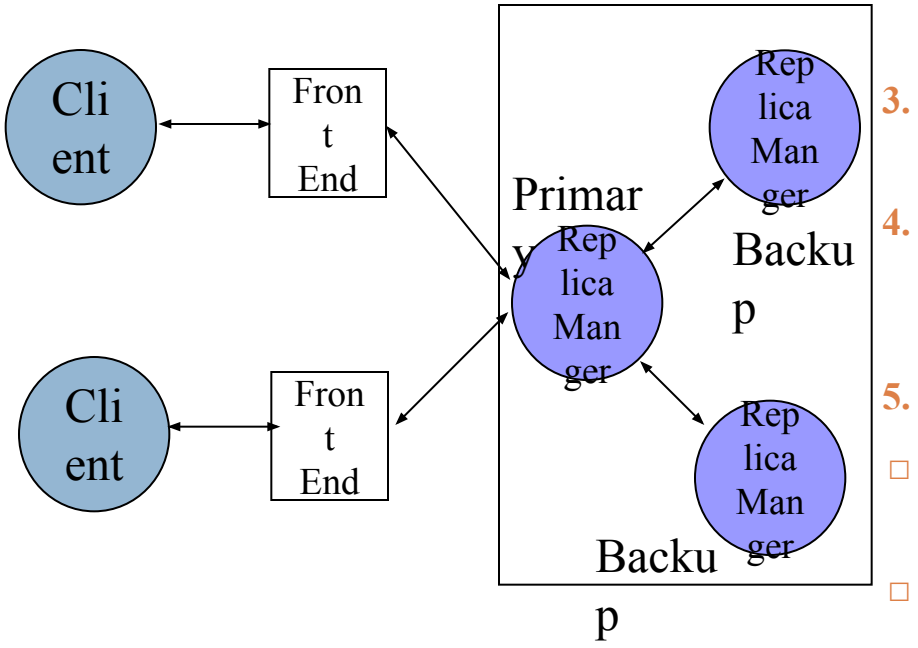
Advantages of Replication

1. Increased Availability
2. Increased Reliability.
3. Improved response time.
4. Reduced network traffic.
5. Improved system throughput.
6. Better scalability.

Multi-copy Update Problem

- Read-only replication
  - Allow the replication of only immutable files.
- Primary backup replication
  - Designate one copy as the primary copy and all the others as secondary copies.
- Active backup replication
  - Access any or all of replicas
    - Read-any-write-all protocol
    - Available-copies protocol
    - Quorum-based consensus

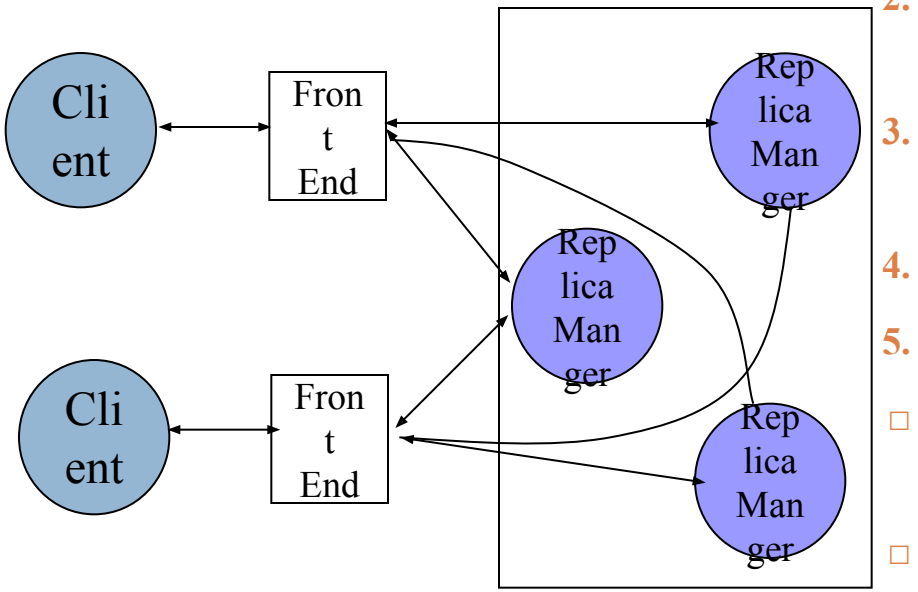
Primary-Copy Replication



1. **Request:** The front end sends a request to the primary replica.
2. **Coordination:**. The primary takes the request atomically.
3. **Execution:** The primary executes and stores the results.
4. **Agreement:** The primary sends the updates to all the backups and receives an ask from them.
5. **Response:** reply to the front end.

- Advantage: an easy implementation, linearizable, coping with n-1 crashes.
- Disadvantage: large overhead especially if the failing primary must be replaced with a backup.

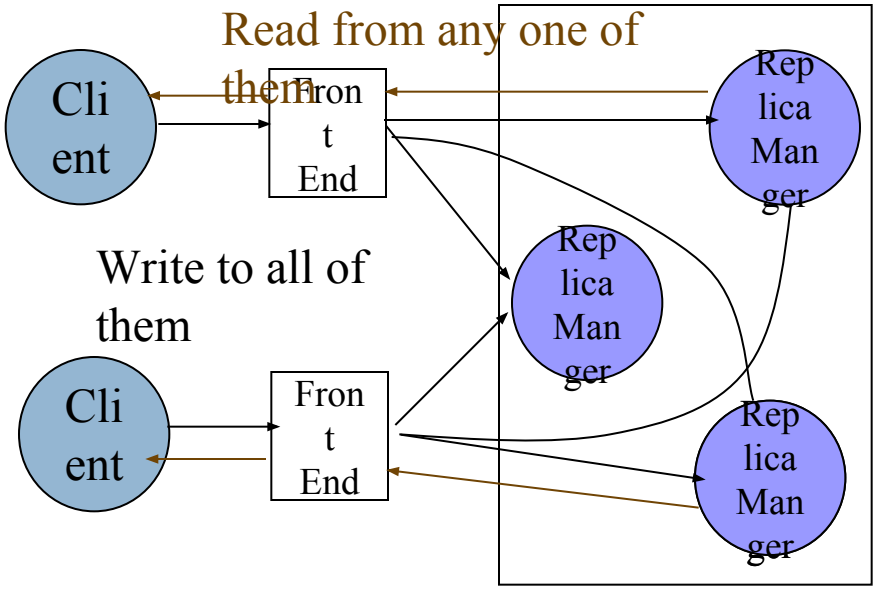
Active Replication



1. **Request:** The front end multicasts to all replicas.
2. **Coordination:**. All replica take the request in the sequential order.
3. **Execution:** Every replica executes the request.
4. **Agreement:** No agreement needed.
5. **Response:** Each replies to the front.

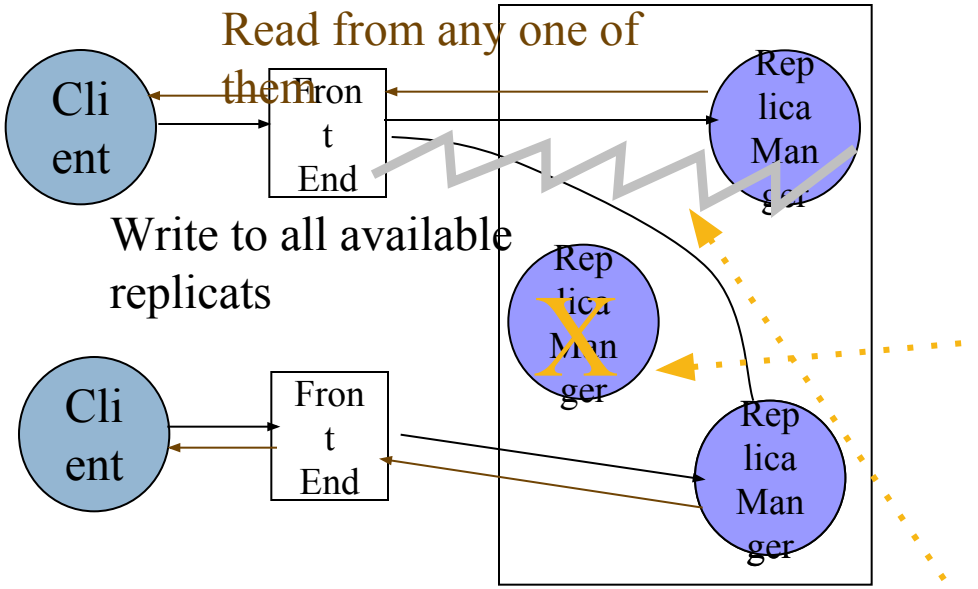
- Advantage: achieve sequential consistency, cope with (n/2 – 1) byzantine failures
- Disadvantage: no more linearizable

Read-Any-Write-All Protocol



- Read
  - Lock any one of replicas for a read
- Write
  - Lock all of replicas for a write
- Sequential consistency
- Intolerable for even 1 failing replica upon a write.

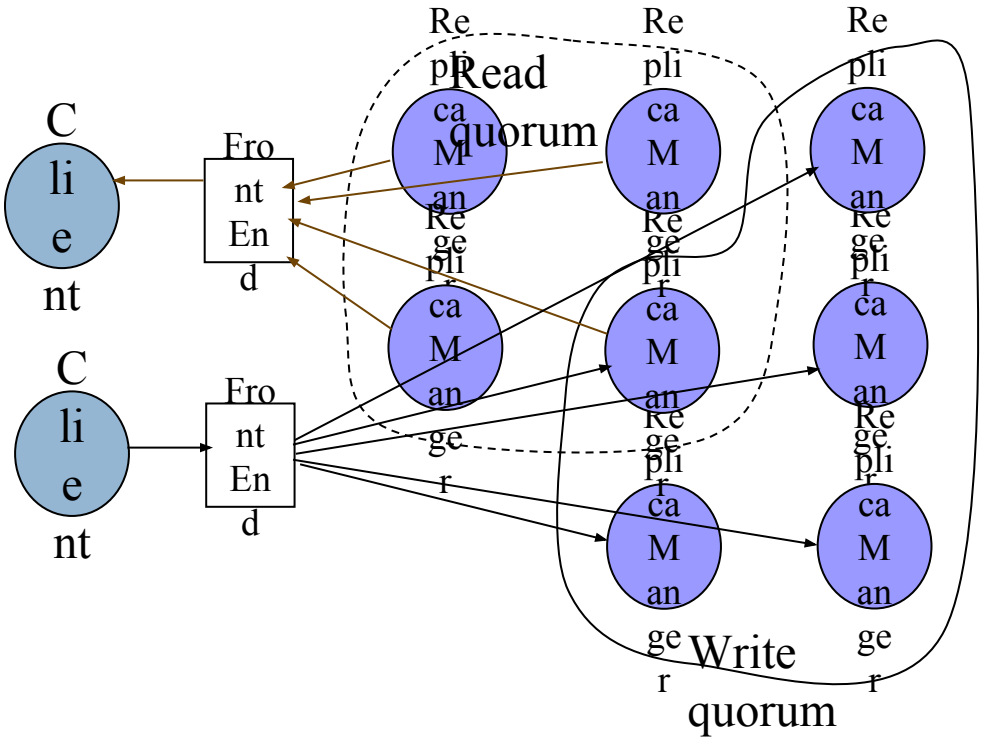
Available-Copies Protocol



- Read
  - Lock any one of replicas for a read
- Write
  - Lock all available replicas for a write
- Recovering replica
  - Bring itself up to date by coping from other servers before accepting any user request.
- Better availability
- Cannot cope with network partition. (Inconsistency in two sub-divided network groups)

Quorum-Based Protocols

#replicas in read quorum + #replicas in write quorum > n



- Read
  - Retrieve the read quorum
  - Select the one with the latest version.
  - Perform a read on it
- Write
  - Retrieve the write quorum.
  - Find the latest version and increment it.
  - Perform a write on the entire write quorum.
- If a sufficient number of replicas from read/write quorum, the operation must be aborted.

Read-any-write-all: r = 1, w = n