Distributed File System : –
File Service Architecture

Vidya Academy of Science & Technology, Thrissur
Sivadasan E T
Associate Professor
Department of CSE.

Distributed File System

- The sharing of resources is a key goal for distributed systems.
- The sharing of <u>stored information</u> is perhaps the most important aspect of distributed resource sharing.

Distributed File System

➢ A distributed file system enables programs to store and access remote files exactly as they do local ones.

➢ Allowing users to access files from any computer on a network.

Distributed File System

- Web servers provide a restricted form of data sharing in which

    - Files stored locally in file systems at the server or in servers on a local network, are made available to clients throughout the Internet.

Distributed File System

- The performance and reliability experienced for access to files stored at a server should be comparable to that for files stored on local disks.

Distributed File System - Challenges

- The design of large-scale wide area read-write file storage systems poses problems of:

  - Load balancing,

  - Reliability,

  - Availability and

  - Security,

Distributed File System - Challenges

- A ***file service*** enables programs to store and access remote files exactly as they do local ones,
  - ➢ Allowing users to access their files from any computer in an intranet.

Distributed File System - Characteristics

- ***File systems*** are responsible for the organization, storage, retrieval, naming, sharing and protection of files.
- They provide a programming interface that characterizes the file abstraction, freeing programmers from concern with the details of storage allocation and layout.

Distributed File System - Characteristics

- ➢ Files are stored on disks or other non-volatile storage media.
- ➢ Files contain both data and attributes.
- ➢ The data consist of a sequence of data items (typically 8-bit bytes), accessible by operations to read and write any portion of the sequence.

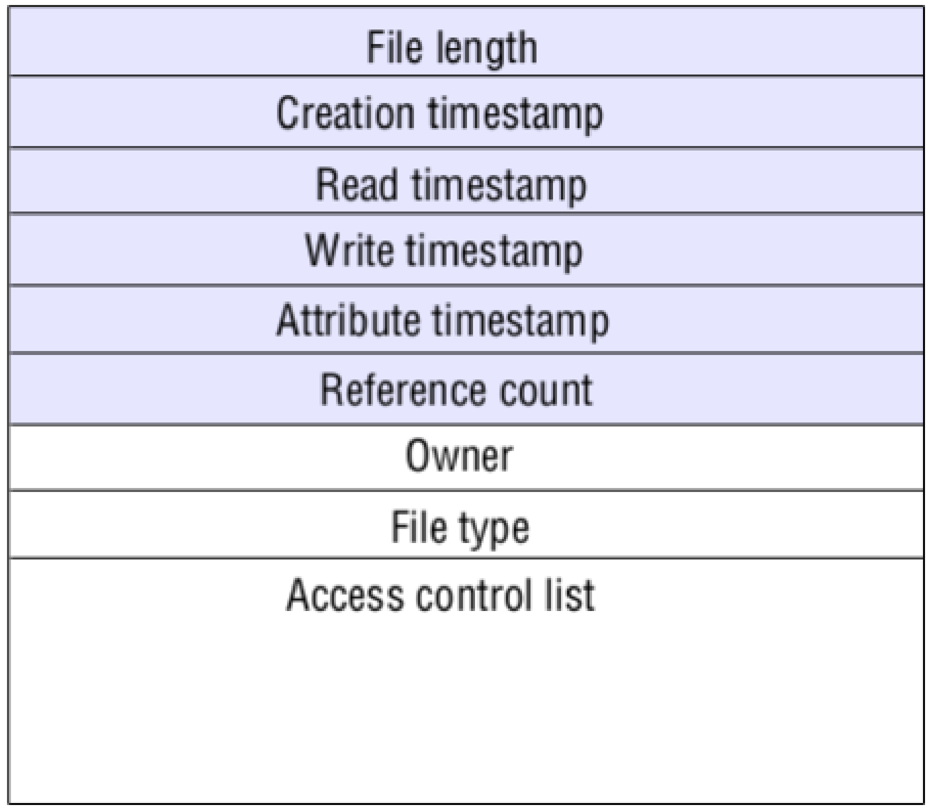
Distributed File System - Characteristics

➢ The attributes are held as a single record containing information such as:

- The length of the file,
- Timestamps,
- File type,
- Owner's identity and
- Access control lists.

Distributed File System - Characteristics

File attribute record structure



A typical attribute record structure is illustrated in Figure. The shaded attributes are managed by the file system and are not normally updatable by user programs.

Distributed File System - Characteristics

- ➢ File systems are designed to store and manage large numbers of files, with facilities for creating, naming and deleting files.
- ➢ The naming of files is supported by the use of directories.

Distributed File System - Characteristics

- Directories may include the names of other directories.
- Leading to the familiar hierarchic filenaming scheme and the multi-part pathnames for files used in UNIX and other operating systems.

Distributed File System - Characteristics

- File systems also take responsibility for:

    1. The control of access to files,

    2.Restricting access to files according to users’ authorizations and

    3. The type of access requested (reading, updating, executing and so on).

Distributed File System - Characteristics

- The main operations on files(create, open, close, read, and write etc...) that are available to applications in UNIX systems.
- These are the system calls implemented by the kernel.

Distributed File System - Characteristics

- ➢ The file system is responsible for applying access control for files.
- ➢ In local file systems such as UNIX, it does so when each file is opened,
- ➢ Checking the rights allowed for the user’s identity in the access control list against the mode of access requested in the open system call.

Distributed File System - Characteristics

- If the rights match the mode,

  - The file is opened and

  - The mode is recorded in the open file state information.

Distributed file system requirements

1. **Access transparency** : -

- Client programs should be unaware of the distribution of files.
- A single set of operations is provided for access to local and remote files.
- Programs written to operate on local files are able to access remote files without modification.

Distributed file system requirements

2. Location transparency :- Client programs should see a uniform file name space.

- Files or groups of files may be relocated without changing their pathnames, and user programs see the same name space wherever they are executed.

Distributed file system requirements

3. **Mobility transparency** - Neither client programs nor system administration tables in client nodes need to be changed when files are moved.

- This allows file mobility – files or, more commonly, sets or volumes of files may be moved, either by system administrators or automatically.

Distributed file system requirements

4. Performance transparency: - Client programs should continue to perform satisfactorily while the load on the service varies within a specified range.

Distributed file system requirements

5. Scaling transparency:- The service can be expanded by incremental growth to deal with a wide range of loads and network sizes.

Distributed file system requirements

6. Concurrent file updates: Changes to a file by one client should not interfere with the operation of other clients simultaneously accessing or changing the same file.

Distributed file system requirements

7. File replication In a file service that supports replication, a file may be represented by several copies of its contents at different locations.

Distributed file system requirements

File replication has two benefits : -

– it enables multiple servers to share the load of providing a service to clients accessing the same set of files and

- It enhances fault tolerance by enabling clients to locate another server that holds a copy of the file when one has failed.

Distributed file system requirements

8. **Hardware and operating system heterogeneity** - The service interfaces should be defined so that client and server software can be implemented for different operating systems and computers.

- This requirement is an important aspect of openness.

Distributed file system requirements

9.Fault tolerance - The central role of the file service in distributed systems makes it essential that the service continue to operate in the face of client and server failures.

Distributed file system requirements

10. Consistency - Conventional file systems such as that provided in UNIX offer one-copy update semantics.

Distributed file system requirements

One-copy update semantics

➢ When files are replicated or cached at

different sites,

➢ There is an inevitable delay in the propagation

of modifications made at one site to all of the

other sites that hold copies,

➢ and this may result in some deviation from

one_copy semantics.

Distributed file system requirements

11. Security - Virtually all file systems provide access-control mechanisms based on the use of access control lists.

- In distributed file systems, there is a need to authenticate client requests.

Distributed file system requirements

Security continues…-

➢ So that access control at the server is based on correct user identities.

➢ Protect the contents of request and reply messages with digital signatures and (optionally) encryption of secret data.

Distributed file system requirements

11. Efficiency - A distributed file service should offer facilities that are of at least the same power.

➢ Generality as those found in conventional file systems.

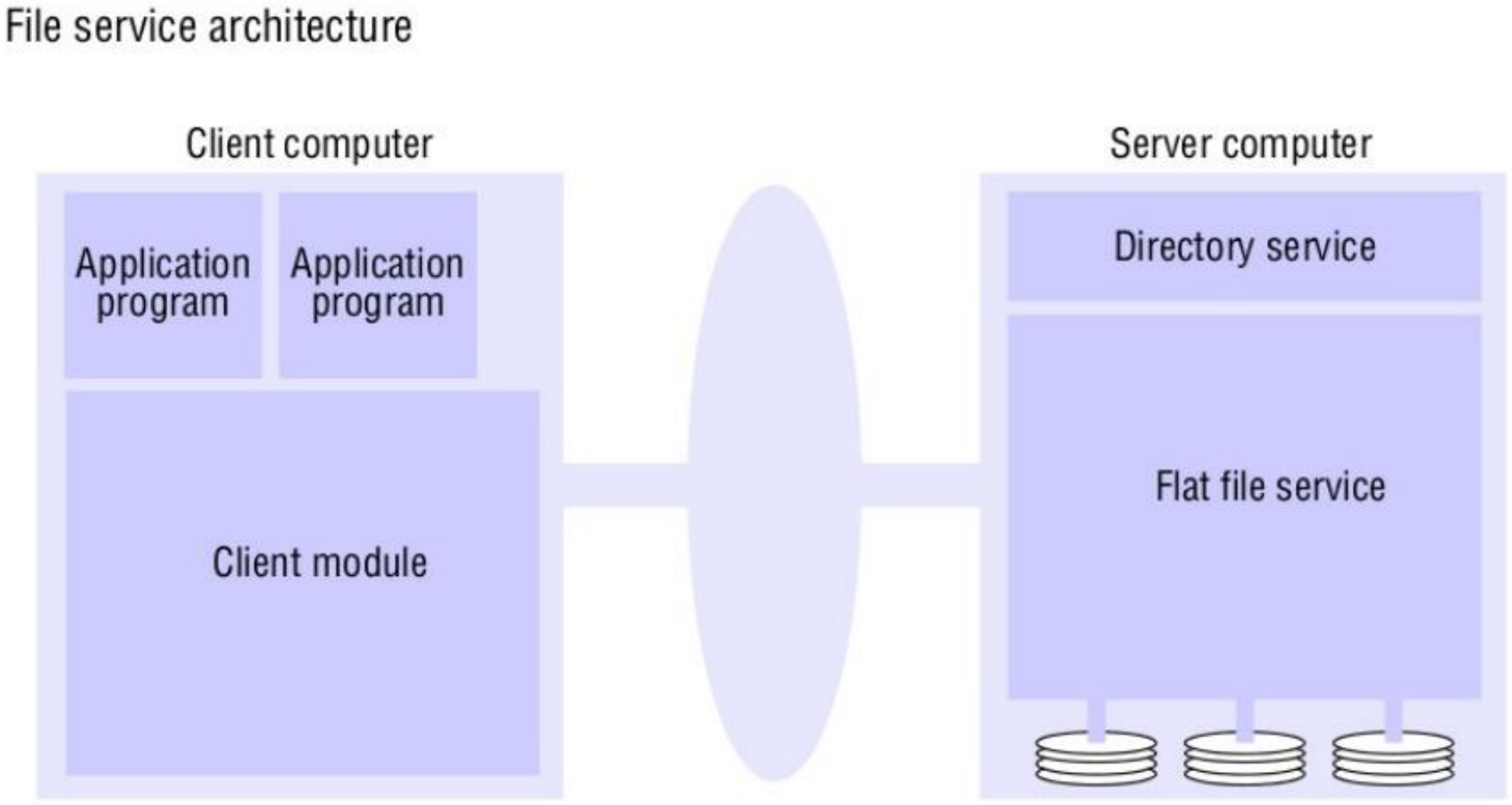
➢ Should achieve a comparable level of performance.

File Service Architecture

➢ An architecture that offers a clear separation of the main concerns in providing access to files is obtained by structuring the file service as three components :-

- Flat file service
- Directory service
- Client module

File Service Architecture



File Service Architecture

The ***flat file service and the directory service***

each export an interface:

➢ For use by client programs, and

➢ Their Request Procedure Call (RPC) interfaces,

➢ Taken together, provide a comprehensive set of

operations for access to files.

File Service Architecture

The ***client module*** provides a single programming interface with operations on files similar to those found in conventional file systems.

File Service Architecture

The ***design is open*** in the sense that :-

- Different client modules can be used to implement different programming interfaces,
- Simulating the file operations of a variety of different operating systems.
- Optimizing the performance for different client and server hardware configurations.

File Service Architecture

Flat file service:

➢ The flat file service is concerned with implementing operations on the contents of files.

➢ Unique file identifiers (UFIDs) are used to refer to files in all requests for flat file service operations.

➢ The division of responsibilities between the file service and the directory service is based upon the use of UFIDs.

File Service Architecture

Flat file service:

➢ UFIDs are long sequences of bits and each file has a UFID that is unique among all of the files in a distributed system.

➢ When the flat file service receives a request to create a file, it generates a new UFID for it and returns the UFID to the requester.

File Service Architecture

Directory service: -

- The *directory service provides a mapping between text names for files and their UFIDs*.

- Clients may obtain the UFID of a file by quoting its text name to the directory service.

File Service Architecture

Directory service: -

➢ The directory service provides the functions needed to generate directories, to add new file names to directories and to obtain UFIDs from directories.

➢ *It is a client of the flat file service*; its directory files are stored in files of the flat file service.

File Service Architecture

Client module : -

➢ A client module runs in each client computer,

➢ Integrating and extending the operations of the flat file service and the directory service under a single application programming interface that is available to user-level programs in client computers.

File Service Architecture

Client module : -

- ➢ The client module also holds information about the network locations of the flat file server and directory server processes.
- ➢ The client module can play an important role in achieving satisfactory performance through the implementation of a cache of recently used file blocks at the client.

File Service Architecture

Flat file service interface

➢ This is the RPC interface used by client modules.

➢ It is not normally used directly by user-level programs.

File Service Architecture

The information below contains a definition of the interface to a flat file service.

; Flat file service operations

Read(FileId, i, n) → Data — throws *BadPosition*	If $1 \le i \le Length(File)$: Reads a sequence of up to *n* items from a file starting at item *i* and returns it in *Data*.
Write(FileId, i, Data) — throws *BadPosition*	If $1 \le i \le Length(File)+1$: Writes a sequence of *Data* to a file, starting at item *i*, extending the file if necessary.
Create() → FileId	Creates a new file of length 0 and delivers a UFID for it.
Delete(FileId)	Removes the file from the file store.
GetAttributes(FileId) → Attr	Returns the file attributes for the file.
SetAttributes(FileId, Attr)	Sets the file attributes (only those attributes that are not shaded in Figure 12.3).

File Service Architecture

Flat file service interface

➢ The most important operations are those for reading and writing.

➢ Both the Read and the Write operation require a parameter ‘i’ specifying a position in the file.

File Service Architecture

Flat file service interface

➢ The Read operation copies the sequence of ‘n’ data items beginning at item ‘i’ from the specified file into Data, which is then returned to the client.

File Service Architecture

Flat file service interface

➢ The Write operation copies the sequence of data items in ‘Data’ into the specified file beginning at item ‘i’,

➢ Replacing the previous contents of the file at the corresponding position and extending the file if necessary.

File Service Architecture

Flat file service interface

➢ Create creates a new, empty file and returns the UFID that is generated.

➢ Delete removes the specified file.

File Service Architecture

Flat file service interface

➢ GetAttributes and SetAttributes enable clients to access the attribute record.

➢ GetAttributes is normally available to any client that is allowed to read the file.

➢ Access to the SetAttributes operation would normally be restricted to the directory service that provides access to the file.

THANK YOU !