



Distributed File Systems

Department of
Computer Science & Engineering

notesforvtu.in

www.cambridge.edu.in



Distributed File Systems

- 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 systems support the sharing of information in the form of files and hardware resources in the form of persistent storage throughout an intranet.
- A well-designed file service provides access to files stored at a server with performance and reliability similar to, and in some cases better than, files stored on local disks.

Storage systems and their properties

Figure 12.1 Storage systems and their properties

	<i>Sharing</i>	<i>Persistence</i>	<i>Distributed cache/replicas</i>	<i>Consistency maintenance</i>	<i>Example</i>
Main memory	×	×	×	1	RAM
File system	×	✓	×	1	UNIX file system
Distributed file system	✓	✓	✓	✓	Sun NFS
Web	✓	✓	✓	×	Web server
Distributed shared memory	✓	×	✓	✓	Ivy (DSM, Ch. 6)
Remote objects (RMI/ORB)	✓	×	×	1	CORBA
Persistent object store	✓	✓	×	1	CORBA Persistent State Service
Peer-to-peer storage system	✓	✓	✓	2	OceanStore (Ch. 10)

Types of consistency:

1: strict one-copy ✓: slightly weaker guarantees 2: considerably weaker guarantees

Characteristics of file systems

- File systems are responsible for the organization, storage, retrieval, naming, sharing and protection of files.
- 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.
 - 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.

File attribute record structure

File length
Creation timestamp
Read timestamp
Write timestamp
Attribute timestamp
Reference count
Owner
File type
Access control list

File System operations

filedes = *open*(*name*, *mode*)

filedes = *creat*(*name*, *mode*)

Opens an existing file with the given *name*.

Creates a new file with the given *name*.

Both operations deliver a file descriptor referencing the open file. The *mode* is *read*, *write* or both.

status = *close*(*filedes*)

Closes the open file *filedes*.

count = *read*(*filedes*, *buffer*, *n*)

count = *write*(*filedes*, *buffer*, *n*)

Transfers *n* bytes from the file referenced by *filedes* to *buffer*.

Transfers *n* bytes to the file referenced by *filedes* from *buffer*. Both operations deliver the number of bytes actually transferred and advance the read-write pointer.

pos = *lseek*(*filedes*, *offset*,
 whence)

Moves the read-write pointer to *offset* (relative or absolute, depending on *whence*).

status = *unlink*(*name*)

Removes the file *name* from the directory structure. If the file has no other names, it is deleted.

status = *link*(*name1*, *name2*)

Adds a new name (*name2*) for a file (*name1*).

status = *stat*(*name*, *buffer*)

Puts the file attributes for file *name* into *buffer*.

Distributed file system requirements

- **Transparency**
 - Access transparency
 - Location transparency
 - Mobility transparency
 - Performance transparency
 - Scaling transparency
- **Concurrent file updates**
- **File replication**

Distributed file system requirements

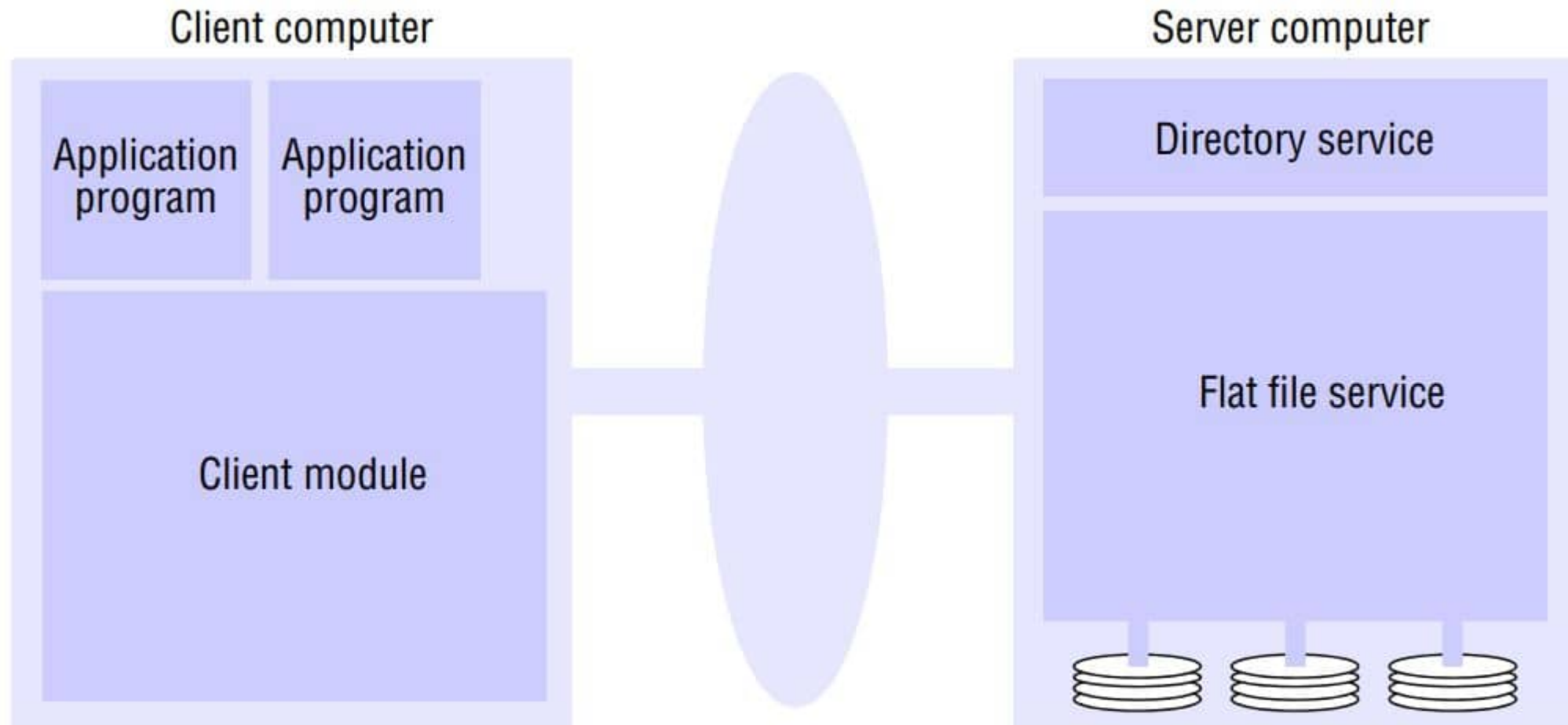
- **Hardware and operating system heterogeneity**
- **Fault tolerance**
- **Consistency**
- **Security**
- **Efficiency**

File Service Architecture

The file service has three components –

- a flat file service,
- a directory service and
- a client module.

File Service Architecture



Flat file service interface

Read(FileId, i, n) → Data
— throws *BadPosition*

If $1 \leq i \leq \text{Length}(\text{File})$: 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 $1 \leq i \leq \text{Length}(\text{File}) + 1$: 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).

Directory service interface

Lookup(Dir, Name) → FileId
— throws *NotFound*

Locates the text name in the directory and returns the relevant UFID. If *Name* is not in the directory, throws an exception.

AddName(Dir, Name, FileId)
— throws *NameDuplicate*

If *Name* is not in the directory, adds (*Name, File*) to the directory and updates the file's attribute record.
If *Name* is already in the directory, throws an exception.

UnName(Dir, Name)
— throws *NotFound*

If *Name* is in the directory, removes the entry containing *Name* from the directory.
If *Name* is not in the directory, throws an exception.

GetNames(Dir, Pattern) → NameSeq

Returns all the text names in the directory that match the regular expression *Pattern*.

Name Services

- Name service is a distinct service that is used by client processes to obtain attributes such as the addresses of resources or objects when given their names.
- The entities named can be of many types, and they may be managed by different services.
- For example, name services are often used to hold the addresses and other details of users, computers, network domains, services and remote objects.

Names, addresses and other attributes

- Any process that requires access to a specific resource must possess a name or an identifier for it.
- The term ***identifier*** is sometimes used to refer to names that are interpreted only by programs.
- Identifiers are chosen for the efficiency with which they can be looked up and stored by software.

Names, addresses and other attributes

- **Pure names** are simply uninterpreted bit patterns.
- **Non-pure names** contain information about the object that they name; in particular, they may contain information about the location of the object.
- **Pure names** always have to be looked up before they can be of any use.
- At the other extreme from a pure name is an object's address: a value that identifies the location of the object rather than the object itself.
- Addresses are efficient for accessing objects, but objects can sometimes be relocated, so addresses are inadequate as a means of identification.

Names and services

- Many of the names used in a distributed system are specific to some particular service.

For example, users of the social networking web site twitter.com, have names such as @magmapoetry that no other service resolves.

- Names are also sometimes needed to refer to entities in a distributed system that are beyond the scope of any single service. The major examples of these entities are users (with proper names and email addresses), computers (with hostnames) and services themselves (such as file service or printer service).

Uniform Resource Identifiers

- Uniform Resource Identifiers (URIs) came about from the need to identify resources on the Web, and other Internet resources such as electronic mailboxes.
- URIs are 'uniform' in that their syntax incorporates that of indefinitely many individual types of resource identifiers (that is, URI schemes), and there are procedures for managing the global namespace of schemes.
- The advantage of uniformity is that it eases the process of introducing new types of identifier, as well as using existing types of identifier in new contexts, without disrupting existing usage.

Uniform Resource Locators

- The familiar term Uniform Resource Locator (URL) is often used for URIs that provide location information and specify the method for accessing the resource.
- For example, `http://www.cdk5.net/` identifies a web page at the given path (‘/’) on the host `www.cdk5.net`, and specifies that the HTTP protocol be used to access it.

Another example is a ‘mailto’ URL, such as `mailto:fred@cambridge.edu.in`, which identifies the mailbox at the given address.

Uniform Resource Locators

- URLs are efficient identifiers for accessing resources. But they suffer from the disadvantage that if a resource is deleted or if it moves, say from one web site to another, there may be dangling links to the resource containing the old URL

Uniform Resource Names

- Uniform Resource Names (URNs) are URIs that are used as pure resource names rather than locators. For example, the URI:
mid:0E4FC272-5C02-11D9-B115-000A95B55BC8@hpl.hp.com

is a URN that identifies the email message containing it in its 'Message-Id' field.

Name Service

- A name service stores information about a collection of textual names, in the form of bindings between the names and the attributes of the entities they denote, such as users, computers, services and objects.
- The collection is often subdivided into one or more naming contexts: individual subsets of the bindings that are managed as a unit.
- The major operation that a name service supports is to resolve a name – that is, to look up attributes from a given name.

General name service requirements

The Global Name Service goals, include:

- To handle an essentially arbitrary number of names and to serve an arbitrary number of administrative organizations.
- A long lifetime.
- High availability.
- Fault isolation.
- Tolerance of mistrust.

Name spaces

A name space is the collection of all valid names recognized by a particular service.

Name spaces require a syntactic definition to separate valid names from invalid names.

For example, '...' is not acceptable as the DNS name of a computer, whereas `www.cdk99.net` is valid.

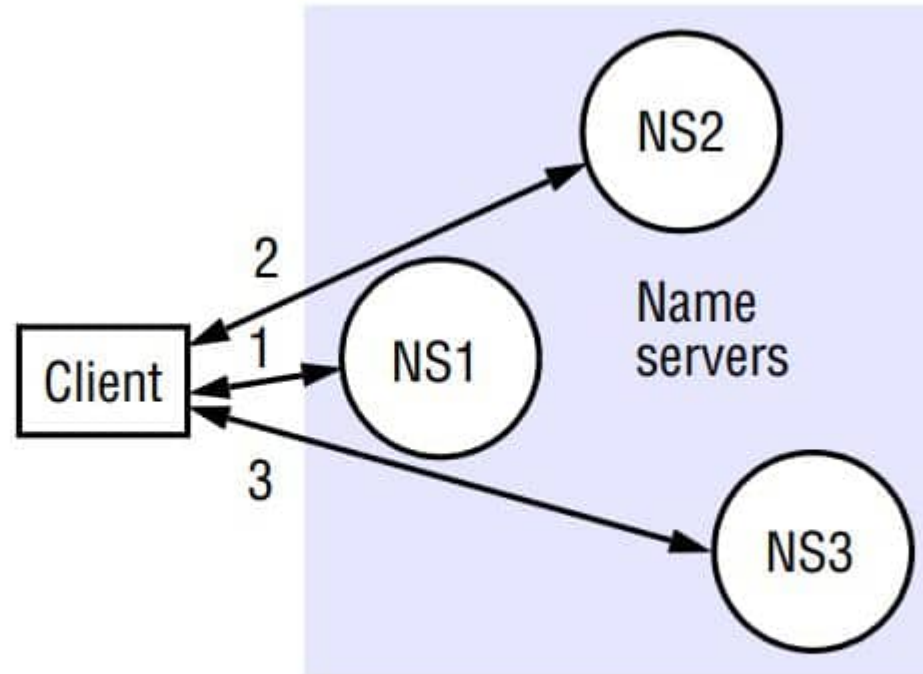
Name servers and Navigation

The process of locating naming data from more than one name server in order to resolve a name is called navigation.

Three navigation models:

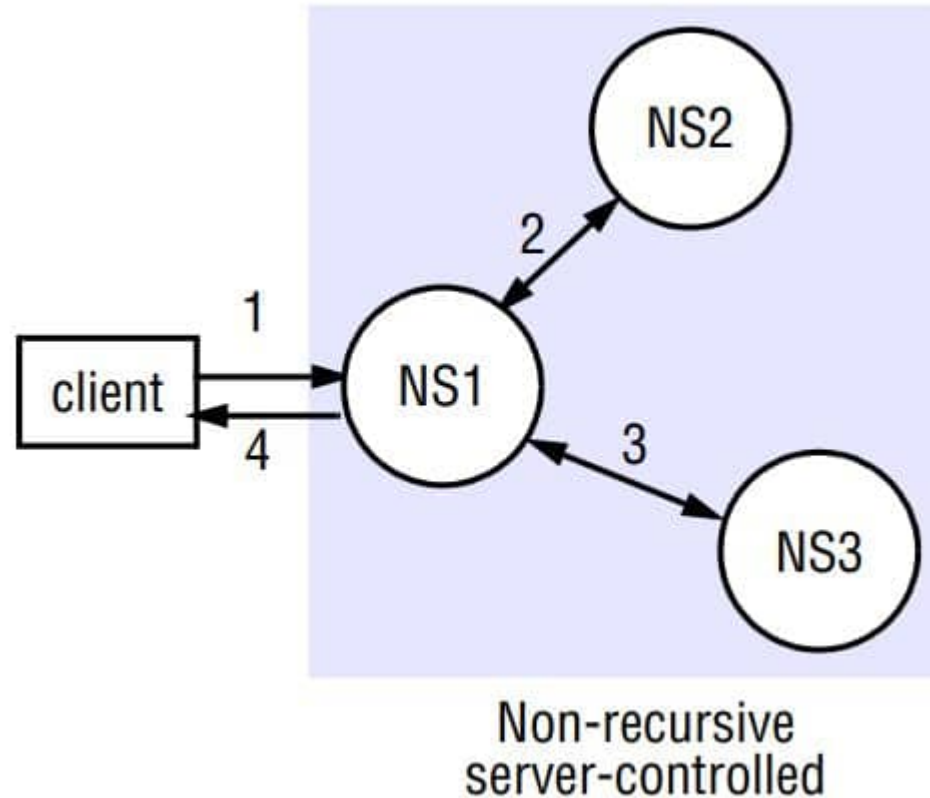
- Iterative Navigation
- Multicast Navigation
- Non-recursive and Recursive Server-controlled Navigation.

Iterative Navigation

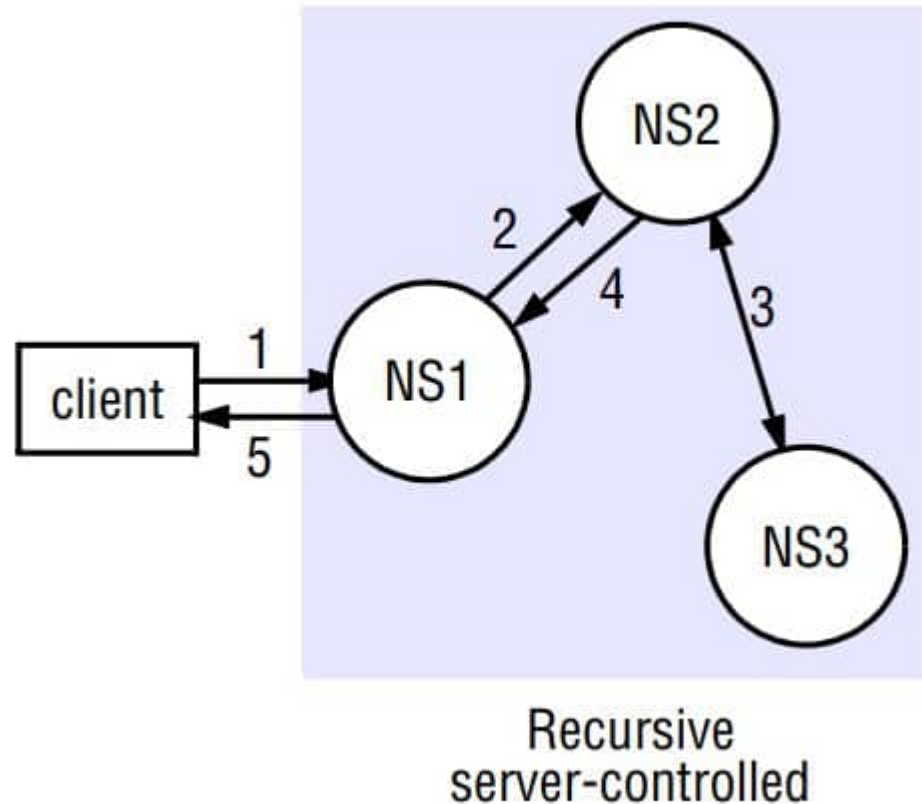


A client iteratively contacts name servers NS1–NS3 in order to resolve a name

Non-recursive Server-controlled Navigation



Recursive Server-controlled Navigation



The Domain Name System

- The Domain Name System is a name service design whose main naming database is used across the Internet.
- It was devised principally by Mockapetris and specified in RFC 1034 [Mockapetris 1987] and RFC 1035.
- DNS replaced the original Internet naming scheme, in which all host names and addresses were held in a single central master file and downloaded by FTP to all computers that required them.

Domain names

The original top-level organizational domains (also called generic domains) in use across the Internet were:

com – Commercial organizations

edu – Universities and other educational institutions

gov – US governmental agencies

mil – US military organizations

net – Major network support centres

org – Organizations not mentioned above

int – International organizations

Domain names

In addition, every country has its own domains:

us – United States

uk – United Kingdom

fr – France

... – ...