Distributed Objects

A distributed object is an object that can be accessed remotely. This means that a distributed object can be used like a regular object, but from anywhere on the network. Distributed objects might be used :

1. to share information across applications or users.
2. to synchronize activity across several machines.
3. to increase performance associated with a particular task.
4. work together by sharing data and invoking methods.
5. This often involves location transparency, where remote objects appear the same as local objects.
6. The main method of distributed object communication is with remote method invocation, generally by message-passing: one object sends a message to another object in a remote machine or process to perform some task. The results are sent back to the calling object.

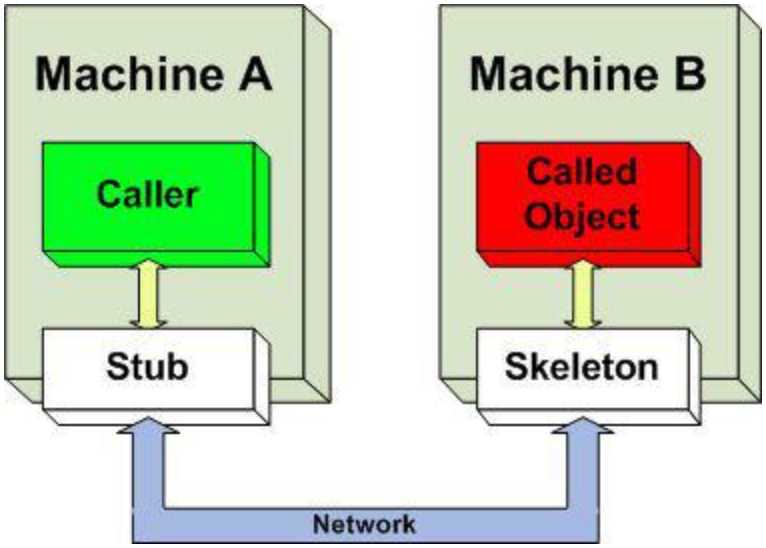
Local Objects vs. Distributed Objects

- Local objects are those whose methods can only be invoked by a local process, a process that runs on the same computer on which the object exists.
- A distributed object is one whose methods can be invoked by a remote process, a process running on a computer connected via a network to the computer on which the object exists.
- Reference : Remote references to distributed objects are more complex than simple pointers to memory addresses
- Request Latency : A distributed object request is orders of magnitude slower than local method invocation
- Object Activation : Distributed objects may not always be available to serve an object request at any point in time
- Parallelism : Distributed objects may be executed in parallel.
- Failure : Distributed objects have far more points of failure than typical local objects.
- Security : Distribution makes them vulnerable to attack.

HOW DISTRIBUTED OBJECT COMMUNICATE ?

The widely used approach on how to implement the communication channel is realized by using stubs and skeletons. They are generated objects whose structure and behavior depends on chosen communication protocol, but in general provide additional functionality that ensures reliable communication over the network.

In RMI, a stub (which is the bit on the client) is defined by the programmer as an interface. The rmic (rmi compiler) uses this to create the class stub. The stub performs type checking. The skeleton is defined in a class which implements the interface stub.



When a caller wants to perform a remote call on the called object, it delegates requests to its stub which initiates communication with the remote skeleton. Consequently, the stub passes caller arguments over the network to the server skeleton. The skeleton then passes received data to the called object, waits for a response and returns the result to the client stub. Note that there is no direct communication between the caller and the called object.

In more details, the communication consists of several steps:

1. caller calls a local procedure implemented by the stub
2. stub marshalls call type and the input arguments into a request message
3. client stub sends the message over the network to the server and blocks the current execution thread
4. server skeleton receives the request message from the network
5. skeleton unpacks call type from the request message and looks up the procedure on the called object
6. skeleton unmarshalls procedure arguments
7. skeleton executes the procedure on the called object
8. called object performs a computation and returns the result
9. skeleton packs the output arguments into a response message
10. skeleton sends the message over the network back to the client
11. client stub receives the response message from the network
12. stub unpacks output arguments from the message
13. stub passes output arguments to the caller, releases execution thread and caller then continues in execution

The advantage of this architecture is that neither the caller nor the called object has to implement network related logic. This functionality, that ensures reliable communication channel over the network, has been moved to the stub and the skeleton layer.

Stub

The client side object participating in distributed object communication is known as a **stub** or **proxy**, and is an example of a proxy object.

The stub acts as a gateway for client side objects and all outgoing requests to server side objects that are routed through it. The stub wraps client object functionality and by adding the network logic ensures the reliable communication channel between client and server. The stub can be written up manually or generated automatically depending on chosen communication protocol.

The stub is responsible for:

- initiating the communication towards the server skeleton
- translating calls from the caller object
- marshalling of the parameters
- informing the skeleton that the call should be invoked
- passing arguments to the skeleton over the network
- unmarshalling of the response from the skeleton
- informing the caller that the call is complete

Skeleton

The server side object participating in distributed object communication is known as a **skeleton** (or stub; term avoided here).

A skeleton acts as gateway for server side objects and all incoming clients requests are routed through it. The skeleton wraps server object functionality and exposes it to the clients, moreover by adding the network logic ensures the reliable communication channel between clients and server. Skeletons can be written up manually or generated automatically depending on chosen communication protocol.

The skeleton is responsible for:

- translating incoming data from the stub to the correct up-calls to server objects
- unmarshalling of the arguments from received data
- passing arguments to server objects
- marshalling of the returned values from server objects
- passing values back to the client stub over the network

RPC

Remote Procedure Call (RPC) is an interprocess communication technique. It is used for client-server applications. RPC mechanisms are used when a computer program causes a procedure or subroutine to execute in a different address space, which is coded as a normal procedure call without the programmer specifically coding the details for the remote interaction. This procedure call also manages low-level transport protocol, such as User Datagram Protocol, Transmission Control Protocol/Internet Protocol etc. It is used for carrying the message data between programs. The Full form of RPC is Remote Procedure Call.

How RPC Works?

RPC architecture has mainly five components of the program:

1. **Client**
2. **Client Stub**
3. **RPC Runtime**
4. **Server Stub**
5. **Server**

Following steps take place during RPC process:

Step 1) The client, the client stub, and one instance of RPC run time execute on the client machine.

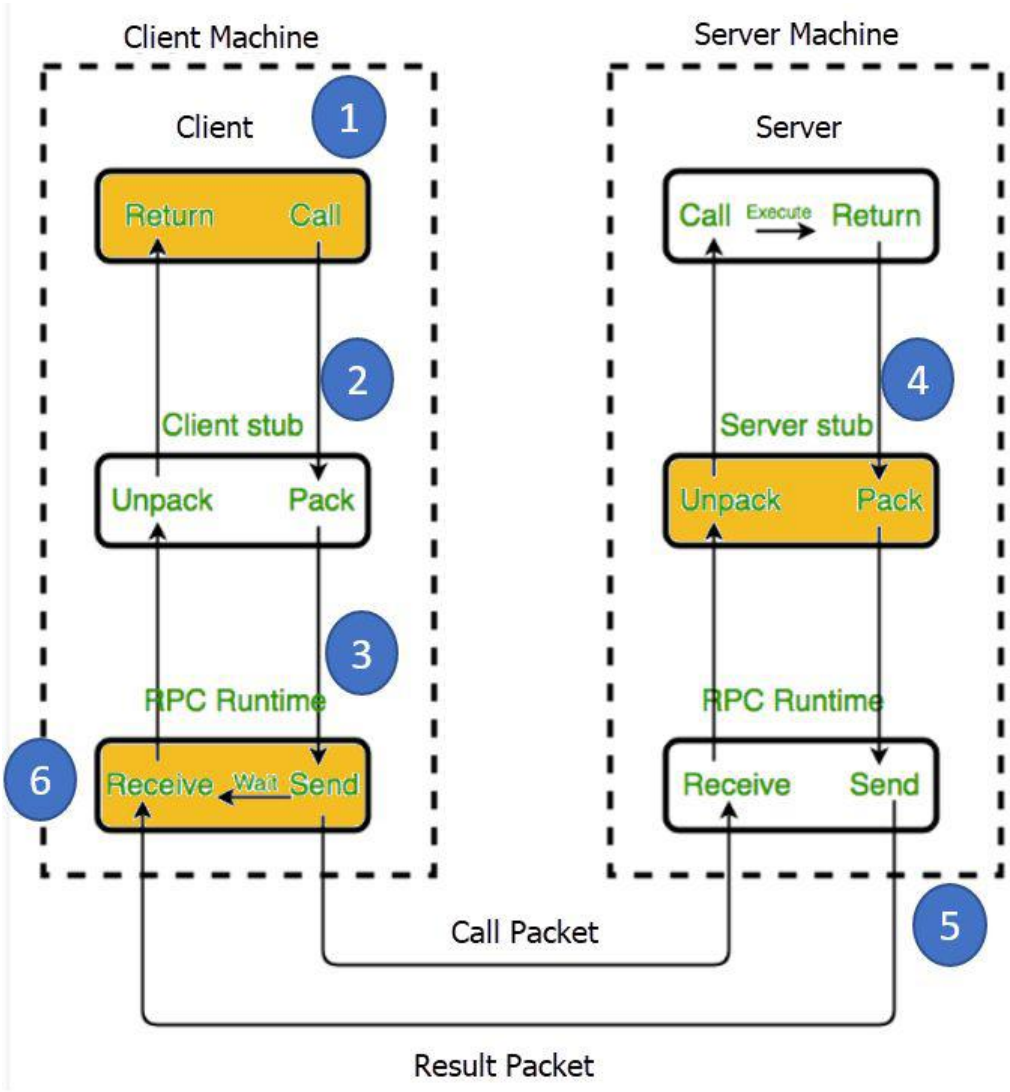
Step 2) A client starts a client stub process by passing parameters in the usual way. The client stub stores within the client's own address space. It also asks the local RPC Runtime to send back to the server stub.

Step 3) In this stage, RPC accessed by the user by making regular Local Procedural Cal. RPC Runtime manages the transmission of messages between the network across client and server. It also performs the job of retransmission, acknowledgment, routing, and encryption.

Step 4) After completing the server procedure, it returns to the server stub, which packs (marshalls) the return values into a message. The server stub then sends a message back to the transport layer.

Step 5) In this step, the transport layer sends back the result message to the client transport layer, which returns back a message to the client stub.

Step 6) In this stage, the client stub demarshalls (unpack) the return parameters, in the resulting packet, and the execution process returns to the caller.



Characteristics of RPC

Here are the essential characteristics of RPC:

- The called procedure is in another process, which is likely to reside in another machine.
- The processes do not share address space.
- Parameters are passed only by values.
- RPC executes within the environment of the server process.
- It doesn't offer access to the calling procedure's environment.

Features of RPC

Here, are some important features of RPC

- Simple call syntax
- Offers known semantics
- Provide a well-defined interface
- It can communicate between processes on the same or different machines

Advantages of RPC

Here, are Pros /benefits of RPC

- RPC method helps clients to communicate with servers by the conventional use of procedure calls in high-level languages.
- RPC method is modeled on the local procedure call, but the called procedure is most likely to be executed in a different process and usually a different computer.
- RPC supports process and thread-oriented models.
- RPC makes the internal message passing mechanism hidden from the user.
- The effort needs to re-write and re-develop the code is minimum.
- Remote procedure calls can be used for the purpose of distributed and the local environment.
- It commits many of the protocol layers to improve performance.
- RPC provides abstraction. For example, the message-passing nature of network communication remains hidden from the user.
- RPC allows the usage of the applications in a distributed environment that is not only in the local environment.
- With RPC code, re-writing and re-developing effort is minimized.
- Process-oriented and thread-oriented models support by RPC.

Disadvantages of RPC

Here, are cons/drawbacks of using RPC:

- Remote Procedure Call Passes Parameters by values only and pointer values, which is not allowed.
- Remote procedure calling (and return) time (i.e., overheads) can be significantly lower than that for a local procedure.
- This mechanism is highly vulnerable to failure as it involves a communication system, another machine, and another process.
- RPC concept can be implemented in different ways, which is can't standard.
- Not offers any flexibility in RPC for hardware architecture as It is mostly interaction-based.

In RPC the caller and callee processes can be situated on different nodes. The normal functioning of an RPC may get disrupted due to one or more reasons mentioned below:

i. Call message is lost or response message is lost

ii. The callee node crashes and is restarted

iii. The caller node crashes and is restarted.

- In RPC system the call semantics determines how often the remote procedure may be executed under fault conditions. The different types of RPC call semantics are as follows:

a. May-Be Call Semantics

- This is the weakest semantics in which a timeout mechanism is used that prevents the caller from waiting indefinitely for a response from the callee.
- This means that the caller waits until a pre-determined timeout period and then continues to execute.
- Hence this semantics does not guarantee the receipt of call message nor the execution. This semantics is applicable where the response message is less important and applications that operate within a local network with successful transmission of messages.

b. Last-Once Call Semantics

- This call semantics uses the idea of retransmitting the call message based on timeouts until the caller receives a response.
- The call, execution and result of will keep repeating until the result of procedure execution is received by the caller.
- The results of the last executed call are used by the caller, hence it known as last-one semantics.
- Last one semantics can be easily achieved only when two nodes are involved in the RPC, but it is tricky to implement it for nested RPCs and cases by orphan calls.

c. Last-of-Many Call Semantics

- This semantics neglects orphan calls unlike last-once call semantics. Orphan call is one whose caller has expired due to node crash.
- To identify each call, unique call identifiers are used which to neglect orphan calls.
- When a call is repeated, it is assigned to a new call identifier and each response message has a corresponding call identifier.
- A response is accepted only if the call identifier associated with it matches the identifier of the most recent call else it is ignored.

d. At-Least-Once Call Semantics

- This semantics guarantees that the call is executed one or more times but does not specify which results are returned to the caller.
- It can be implemented using timeout based retransmission without considering the orphan calls.

e. Exactly-Once Call Semantics

- This is the strongest and the most desirable call semantics. It eliminates the possibility of a procedure being executed more than once irrespective of the number of retransmitted call.
- The implementation of exactly-once call semantics is based on the use of timeouts, retransmission, call identifiers with the same identifier for repeated calls and a reply cache associated with the callee.

Semantics of RPC

Unlike normal procedure calls, many things can go wrong with RPC. Normally, a client will send a request, the server will execute the request and then return a response to the client. What are appropriate semantics for server or network failures? Possibilities:

1. Just hang forever waiting for the reply that will never come. This models regular procedure call. If a normal procedure goes into an infinite loop, the caller never finds out. Of course, few users will like such semantics.
2. Time out and raise an exception or report failure to the client. Of course, finding an appropriate timer value is difficult. If the remote procedure takes a long time to execute, a timer might time-out too quickly.
3. Time out and retransmit the request.

While the last possibility seems the most reasonable, it may lead to problems. Suppose that:

1. The client transmits a request, the server executes it, but then crashes before sending a response. If we don't get a response, is there any way of knowing whether the server acted on the request?
2. The server restarts, and the client retransmits the request. What happens? Now, the server will reject the retransmission because the supplied unique identifier no longer matches that in the server's export table. At this point, the client can decide to rebind to a new server and retry, or it can give up.
3. Suppose the client rebinds to the another server, retransmits the request, and gets a response. How many times will the request have been executed? At least once, and possibly twice. We have no way of knowing.

Operations that can safely be executed twice are called *idempotent*. For example, fetching the current time and date, or retrieving a particular page of a file. Is deducting $10,000 from an account idempotent? No. One can only deduct the money once. Likewise, deleting a file is not idempotent. If the delete request is executed twice, the first attempt will be successful, while the second attempt produces a ``nonexistent file'' error.

IDL

Interface Definition Language (IDL) is a standard language for defining function and method interfaces for distributed or component-based applications. Interface Description Language is a purely declarative language that specifies the interface that an object should implement and a client must use.

IDL (interface definition language) is a generic term for a language that lets a program or object written in one language communicate with another program written in an unknown language. In distributed object technology, it's important that new objects be able to be sent to any platform environment and discover how to run in that environment.

RMI

he **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects: stub and *skeleton*.

stub

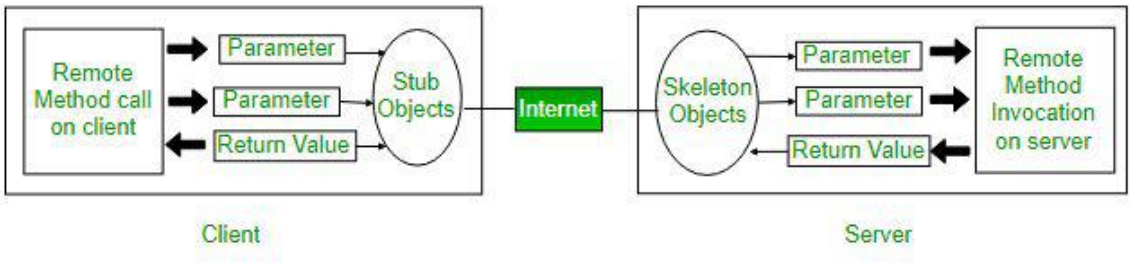
The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.



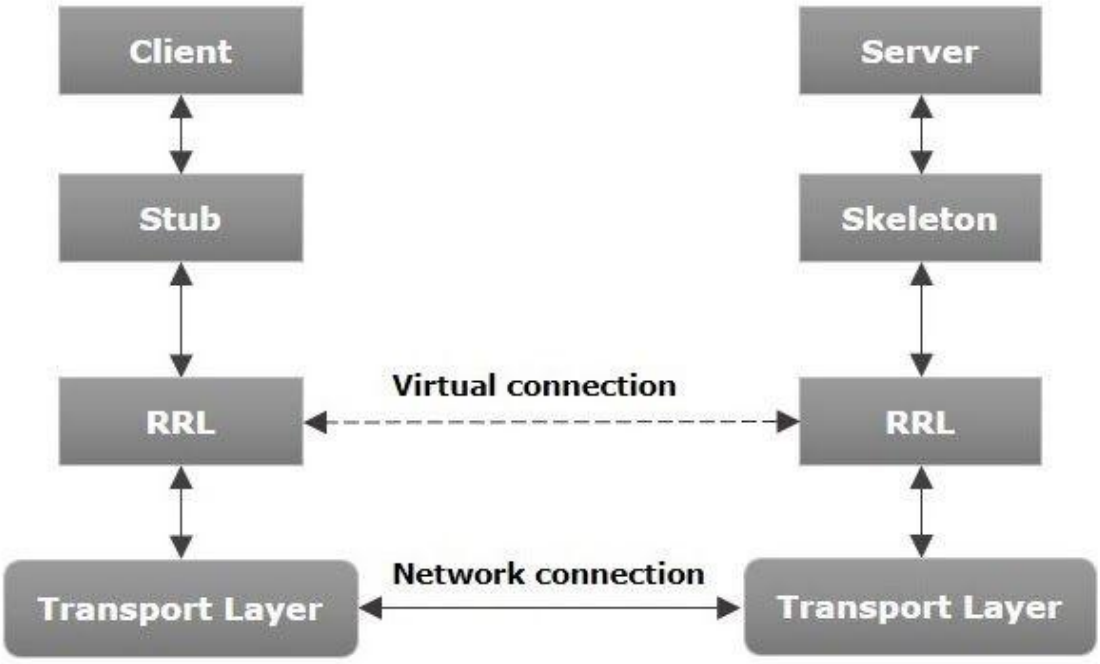
Working of RMI

Architecture of an RMI Application

In an RMI application, we write two programs, a **server program** (resides on the server) and a **client program** (resides on the client).

- Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).
- The client program requests the remote objects on the server and tries to invoke its methods.

The following diagram shows the architecture of an RMI application.



Let us

now discuss the components of this architecture.

- **Transport Layer** − This layer connects the client and the server. It manages the existing connection and also sets up new connections.
- **Stub** − A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.
- **Skeleton** − This is the object which resides on the server side. **stub** communicates with this skeleton to pass request to the remote object.
- **RRL(Remote Reference Layer)** − It is the layer which manages the references made by the client to the remote object.

Working of an RMI Application

The following points summarize how an RMI application works −

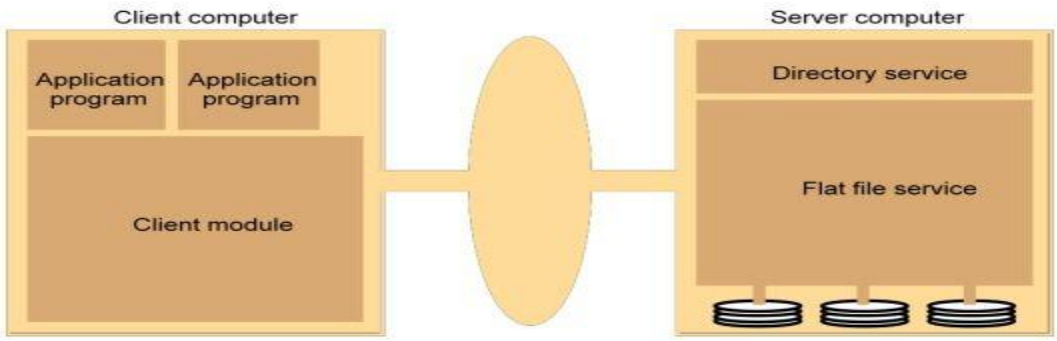
- When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the RRL.
- When the client-side RRL receives the request, it invokes a method called **invoke()** of the object **remoteRef**. It passes the request to the RRL on the server side.
- The RRL on the server side passes the request to the Skeleton (proxy on the server) which finally invokes the required object on the server.
- The result is passed all the way back to the client.

RPC	RMI
RPC is a library and OS dependent platform.	Whereas it is a java platform.
RPC supports procedural programming.	RMI supports object oriented programming.
RPC is less efficient in comparison of RMI.	While RMI is more efficient than RPC.
RPC creates more overhead.	While it creates less overhead than RPC.
The parameters which are passed in RPC are ordinary or normal data.	While in RMI, objects are passed as parameter.
RPC is the older version of RMI.	While it is the successor version of RPC.
There is high Provision of ease of programming in RPC.	While there is low Provision of ease of programming in RMI.
RPC does not provide any security.	While it provides client level security.
It's development cost is huge.	While it's development cost is fair or reasonable.
There is a huge problem of versioning in RPC.	While there is possible versioning using RDMI.
There is multiple codes are needed for simple application in RPC.	While there is multiple codes are not needed for simple application in RMI.

Distributed File System

A distributed file system (DFS) is a file system with data stored on a server. The data is accessed and processed as if it was stored on the local client machine. The DFS makes it convenient to share information and files among users on a network in a controlled and authorized way. The server allows the client users to share files and store data just like they are storing the information locally. However, the servers have full control over the data and give access control to the clients.

One process involved in implementing the DFS is giving access control and storage management controls to the client system in a centralized way, managed by the servers. Transparency is one of the core processes in DFS, so files are accessed, stored, and managed on the local client machines while the process itself is actually held on the servers. This transparency brings convenience to the end user on a client machine because the network file system efficiently manages all the processes. Generally, a DFS is used in a LAN, but it can be used in a WAN or over the Internet.



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:

1. A flat file service
2. A directory service
3. A client module.

The Client module implements exported interfaces by flat file and directory services on server side. Responsibilities of various modules can be defined as follows:

- **Flat file service** : Concerned with the implementation of operations on the contents of file. Unique File Identifiers (UFIDs) are used to refer to files in all requests for flat file service operations. UFIDs are long sequences of bits chosen so that each file has a unique among all of the files in a distributed system.
- **Directory service:** Provides mapping between text names for the files and their UFIDs. Clients may obtain the UFID of a file by quoting its text name to directory service. Directory service supports functions needed to generate directories, to add new files to directories.

- **Client Module:** It runs on each computer and provides integrated service (flat file and directory) as a single API to application programs. For example, in UNIX hosts, a client module emulates the full set of Unix file operations. It holds information about the network locations of flat-file and directory server processes; and achieve better performance through implementation of a cache of recently used file blocks at the client.
- **Access control:** In distributed implementations, access rights checks have to be performed at the server because the server RPC interface is an otherwise unprotected point of access to files.
- Flat file service RPC interface
  • Used by client modules, not user programs
    – FileId (UFID) uniquely identifies file
    – invalid if file not present or inappropriate access
    – Read/Write; Create/Delete; Get/SetAttributes
  • No open/close! (unlike UNIX)
    – access immediate with FileId
    – Read/Write identify starting point
  • Improved fault-tolerance
    – operations idempotent except Create, can be repeated (atleast-once RPC semantics)
    – stateless service

[source: https://www.brainkart.com/article/File-Service-Architecture_8545/]

Distributed File Systems: Issues

- Naming and Transparency
- Remote file access
- Caching
- Server with state or without
- Replication

Advantages:

- Access time reduced.
- Safer if node fails.
- Does not require client nodes to have large storage space.

Disadvantages:

- Difficult to keep local copy consistent with remote copy.
- Slower than just keeping it in local memory.

[http://www.cse.chalmers.se/edu/year/2010/course/EDA092/SLIDES/15-distributed-filesystems.pdf]

Related requirements in distributed file systems are:

1. Transparency.Transparency (clients unaware of the distributed nature)
   – access transparency (client unaware of distribution of files, same interface for local/remote files)
   – location transparency (uniform file name space from any client workstation)
   – mobility transparency (files can be moved from one server to another without affecting client)
   – performance transparency (client performance not affected by load on service)
   – scaling transparency
2. Concurrency: Concurrent file updates (changes by one client do not affect another)
3. Replication: File replication (for load sharing, fault-tolerance)
4. Heterogeneity : (interface platform-independent)
5. Fault tolerance: (continues to operate in the face of client and server failures)
6. Consistency : (one-copy-update semantics or slight variations)
7. Security : (access control)
8. Efficiency :(performance comparable to conventional file systems)

File Service :

One of the most important services provided by a distributed system is file service as other services depend on file service (print service, naming service). Users contact the distributed system through file service.

Basic aspects of file services and servers

Servers : service software running on single machine
Services: Software entity running on one or more machine

Service provided by file service

- Sharing data and database (automatic access, backup and recovery
- User mobility (user may need to use different computers at different times)
- Low cost solution, can be used in extreme environment

[http://mazsola.iit.uni-miskolc.hu/DATA/research/tempus/discom/doc/os/obsolete/dfile.ps.gz]

Directory Services

A directory service is a customizable information store that functions as a single point from which users can locate resources and services distributed throughout the network. This customizable information store also gives administrators a single point for managing its objects and their attributes. Although this information store appears as a single point to the users of the network, it is actually most often stored in a distributed form.

Directory services are network services that identify every resource such as email address, peripheral devices and computers on the network, and make these resources accessible to users and applications.

Specific directory services called naming services map the names of resources in the network to the respective network address. This directory service relieves users from having to know the physical addresses of network resources. Directory services also define namespaces for networks, which hold one or more objects as name entries.

Directory services hold shared information infrastructure to administer, manage, locate and organize common items and network resources. It is also a vital component of network operating systems.

SUN's NFS

Network File System (NFS) is a distributed file system protocol originally developed by Sun Microsystems (Sun) in 1984, allowing a user on a client computer to access files over a computer network much like local storage is accessed.

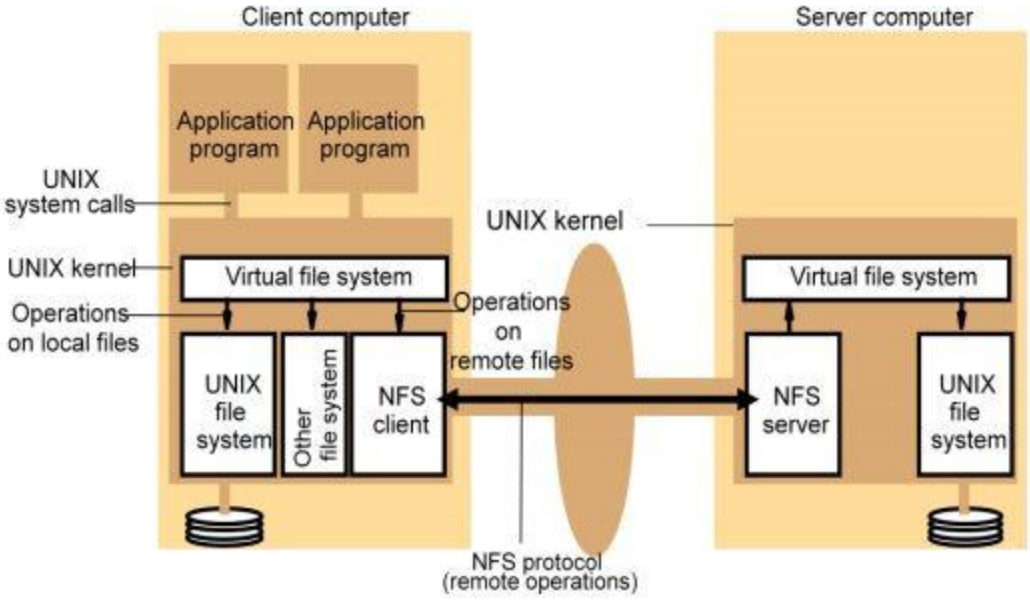
The earliest successful distributed system could be attributed to Sun Microsystems, which developed the Network File System (NFS). NFSv2 was the standard protocol followed for many years, designed with the goal of simple and fast server crash recovery. This goal is of utmost importance in multi-client and single-server based network architectures because a single instant of server crash means that all clients are unserviced. The entire system goes down.

Stateful protocols make things complicated when it comes to crashes. Consider a client A trying to access some data from the server. However, just after the first read, the server crashed. Now, when the server is up and running, client A issues the second read request. However, the server does not know which file the client is referring to, since all that information was temporary and lost during the crash.

Stateless protocols come to our rescue. Such protocols are designed so as to not store any state information in the server. The server is unaware of what the clients are doing — what blocks they are caching, which files are opened by them and where their current file pointers are. The server simply delivers all the information that is required to service a client request. If a server crash happens, the client would simply have to retry the request. Because of their simplicity, NFS implements a stateless protocol.

Stateless Protocol	Stateful Protocol
Stateless Protocol does not require the server to retain the server information or session details.	Stateful Protocol require server to save the status and session information.
In Stateless Protocol, there is no tight dependency between server and client.	In Stateful protocol, there is tight dependency between server and client
The Stateless protocol design simplify the server design.	The Stateful protocol design makes the design of server very complex and heavy.
Stateless Protocols works better at the time of crash because there is no state that must be restored, a failed server can simply restart after a crash.	Stateful Protocol does not work better at the time of crash because stateful server have to keep the information of the status and session details of the internal states.
Stateless Protocols handle the transaction very fastly.	Stateful Protocols handle the transaction very slowly.
Stateless Protocols are easy to implement in Internet.	Stateful protocols are logically heavy to implement in Internet.

- The NFS server is a stateless server, so the user's identity and access rights must be checked by the server on each request.
- In the local file system they are checked only on the file's access permission attribute.
- Every client request is accompanied by the userID and groupID
- Kerberos has been integrated with NFS to provide a stronger and more comprehensive security solution.
- Server maintains a table of clients who have mounted filesystems at that server.
- Each client maintains a table of mounted file systems holding
- Remote file systems may be hard-mounted or soft-mounted in a client computer.



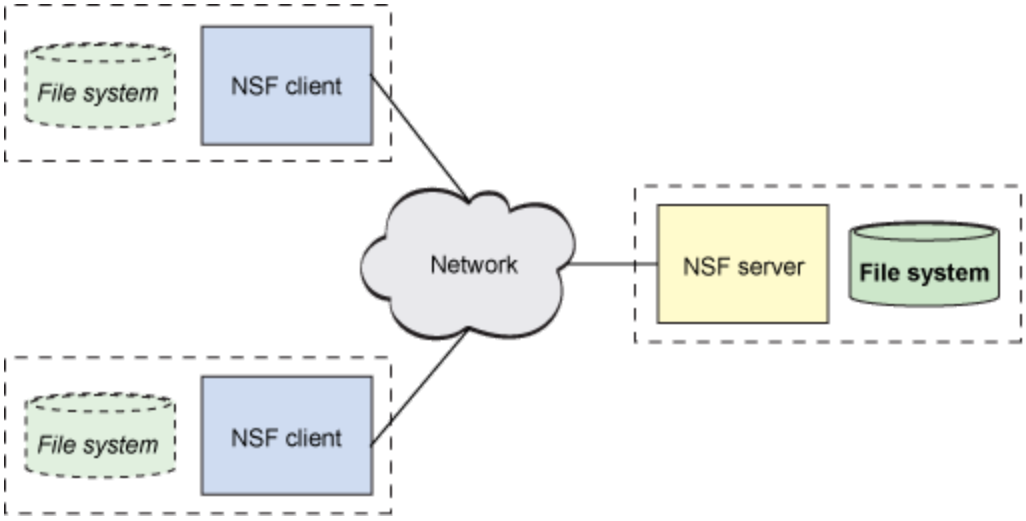
One computer will be the NFS server: it offers up various chunks of disk storage (which is usually directly attached to the server system) to NFS clients that it can exchange TCP/IP packets with.

The other computer will be the NFS client: it will make a "mount" request of the NFS server, and subsequently make open/read/write/lseek/close requests of the NFS server on a per-file basis.

A network file system (NFS) is a type of file system mechanism that enables the storage and retrieval of data from multiple disks and directories across a shared network.

A network file system enables local users to access remote data and files in the same way they are accessed locally.

NFS was initially developed by Sun Microsystems.



NFS is derived from the distributed file system mechanism. It is generally implemented in computing environments where the centralized management of data and resources is critical. Network file system works on all IP-based networks. It uses TCP and UDP for data access and delivery, depending on the version in use.

The network file system is implemented in a client/server computing model, where an NFS server manages the authentication, authorization, and management of clients, as well as all the data shared within a specific file system. Once authorized, clients can view and access the data through their local systems much like they'd access it from an internal disk drive.

Naming In Distributed System

Names play a very important role in all computer systems. They are used to share resources, to uniquely identify entities, to refer to locations, and more. An important issue with naming is that a name can be resolved to the entity it refers to. Name resolution thus allows a process to access the named entity. To resolve names, it is necessary to implement a naming system. The difference between naming in distributed systems and non distributed systems lies in the way naming systems are implemented.

In a distributed system, the implementation of a naming system is itself often distributed across multiple machines. How this distribution is done plays a key role in the efficiency and scalability of the naming system. In this chapter, we concentrate on three different, important ways that names are used in distributed systems.

Domain Name System (DNS)

The domain name system (DNS) is a naming database in which internet domain names are located and translated into internet protocol (IP) addresses. The domain name system maps the name people use to locate a website to the IP address that a computer uses to locate a website. For example, if someone types example.com into a web browser, a server behind the scenes will map that name to the corresponding IP address, something similar in structure to 121.12.12.121.

How DNS work ???

DNS servers answer questions from both inside and outside their own domains. When a server receives a request from outside the domain for information about a name or address inside the domain, it provides the authoritative answer. When a server receives a request from inside its own domain for information about a name or address outside that domain, it passes the request out to another server. Usually, this server is one managed by its internet service provider (ISP). If that server does not know the answer or the authoritative source for the answer, it will reach out to the DNS servers for the top-level domain -- e.g., for all of .com or .edu. Then, it will pass the request down to the authoritative server for the specific domain -- e.g., itsaihub.com or aihubprojects.com

DNS structure

A domain name is made of multiple parts, called labels. The domain hierarchy is read from right to left with each section denoting a subdivision. The top-level domain is what appears after the period in the domain name. A few examples of top-level domains are .com, .org and .edu, but there are many others that can be used. Some may denote a country code or geographic location such as .us for the United States or .ca for Canada.

Each label to the left denotes another subdomain to the right. So for example, “techtarget” is a subdomain of .com. and “www.” is a subdomain of techtarget.com. There can be up to 127 levels of subdomains, and each label can have up to 63 characters. The total domain character length can have up to 253 characters. Other rules include not starting or ending labels with hyphens and not having a fully numeric top-level domain name.

DNS (Domain Name System)

DNS stands for Domain Name Service , It acts as a look-up table which allows the correct servers to be contacted when the user enters the URL into the web browser , This transparent service offers the other features which are commonly used by the webmasters to organize their data infrastructure .

DNS runs on DNS servers , When the user enters the URL , such as www.google.com , into the Web browser , The request is not directly sent to the Google servers , Instead , the request goes to the DNS server , that uses a look-up table to determine several pieces of information , IP address of the website that is being requested , Then it forwards this request to the proper servers & returns the information requested to the user's web browser .

DNS is a hierarchical naming system for the computer systems , the services or for any resources participating in the internet , Many information with domain names is assigned to each of the participants , It can translate the names of the domain into the binary identifiers that are associated with the equipment of the network to locate & address these devices .

DNS advantages

Domain Name System (DNS) is a centralized mechanism for resolving / giving the IP addresses for a given domain name , It is the system that helps you to find the website using your internet browser , When you click on your internet browser (Internet Explorer , Safari , Firefox etc.) , You will be able to type the name of the website .

It is the only system of its kind that will allow you to browse & use the internet , Its use is necessary for most companies & the people across the world , Without this system it would be impossible for the people to access the internet & the internet has become an essential part of our society .

You do not need to memorize numbers , The domain names make / give a kind of sense to hyper the links when the name is given instead of a string of numbers , It is easy for categorizing , archiving & helping the search engines .

DNS enables you to specify the technical functionality of the database service , It can define the DNS protocol , the detailed specification of the data structures & the data communication exchanges used in the DNS , DNS is used as a form of load balancing or an additional layer of security .

Host-names & IP addresses are not required to match in a one-to-one relationship , Multiple host-names may correspond to a single IP address , that is useful in virtual hosting , in which many web sites are served from a single host , a single host-name may resolve to many IP addresses to facilitate fault tolerance & load distribution to multiple server instances across an enterprise or the global Internet .

DNS can enhance the security of your DNS infrastructure , It can allow the dynamic secure updates , It is more reliable , It can deliver the messages to the users with zero downtime , It is faster , It is connected well at intersections of internet , It enables the requests to be answered to the next closest node in the case of maintenance or downtime , It is smarter & It offers automatic corrections of typos .

DNS disadvantages

The hierarchical & centralized which breaks down the main objective of the Internet is designed to be a decentralized system , ICANN can control the DNS root registry that is a non profit private organization with ties to one specific nation & challenges the concept of net neutrality .

Breakdown of Domain Name System will crash the world wide web although there are many root servers & backup servers targeting DNS servers at particular key locations will do a lot of harm , Spoofing DNS will lead to a lot of crucial/private data ending up in wrong hands .

If the server or computer breaks then the web-page hosted by the server or computer cannot run , DNS issues can be difficult to troubleshoot due to its geographical & distributed nature , The clients can not connect to their local network when DNS is broken and they will be unable to reach the remote devices by their names .

DNS queries do not carry any information about the client that triggered the name resolution , The service-side DNS server knows only the network address of the DNS server that asks about the service location .

When the DNS server can not find the correct IP address , The website will not load , As the computers communicate via IP addresses & not host-names , The computer doesn't know what you're trying to reach unless it can use an IP address .

If the malware changed your DNS server settings , entering the same URL might take you to a completely different website or to the website that looks like your bank website but really isn't , It may record your username & password , giving the scammers all the information they need to access your bank account .

Malwares hijack some DNS servers to redirect the popular websites to ones that are full of the advertisements or fake virus websites that make you think you have to buy the program to clean the infected computer .

Issue	NFS	Coda	Plan 9	xFS	SFS
Design goals	Access transparency	High availability	Uniformity	Serverless system	Scalable security
Access model	Remote	Up/Download	Remote	Log-based	Remote
Communication	RPC	RPC	Special	Active msgs	RPC
Client process	Thin/Fat	Fat	Thin	Fat	Medium
Server groups	No	Yes	No	Yes	No
Mount granularity	Directory	File system	File system	File system	Directory
Name space	Per client	Global	Per process	Global	Global
File ID scope	File server	Global	Server	Global	File system
Sharing sem.	Session	Transactional	UNIX	UNIX	N/S
Cache consist.	write-back	write-back	write-through	write-back	write-back
Replication	Minimal	ROWA	None	Striping	None
Fault tolerance	Reliable comm.	Replication and caching	Reliable comm.	Striping	Reliable comm.
Recovery	Client-based	Reintegration	N/S	Checkpoint & write logs	N/S
Secure channels	Existing mechanisms	Needham-Schroeder	Needham-Schroeder	No pathnames	Self-cert.
Access control	Many operations	Directory operations	UNIX based	UNIX based	NFS BASED

A comparison between NFS, Coda, Plan 9, xFS. N/S indicates that nothing has been specified.