KARACHI INSTITUTE OF ECONOMICS & TECHNOLOGY

Parallel & Distributed Computing Systems (*PNDC*)

Project Report

Teacher Name: Miss Umme Kulsoom

CLASS ID: 108980

Client To Server Chat Application In C#

Abstract:

Chat refers to the process of communicating, interacting

and/or exchanging messages over the Internet. It involves two

or more individuals that communicate through a chat-enabled

service or software. Chat may be delivered through text, audio

or video communication via the Internet.

A chat application has basic two components, viz server and

client. A server is a computer program or a device that

provides functionality for other programs or devices. Clients

who want to chat with each other connect to the server.

The chat application we are going to make will be more like a

chat room, rather than a peer to peer chat. So this means that

multiple users can connect to the chat server and send their

messages. Every message is broadcasted to every connected

chat user.

TABLE OF CONTENTS

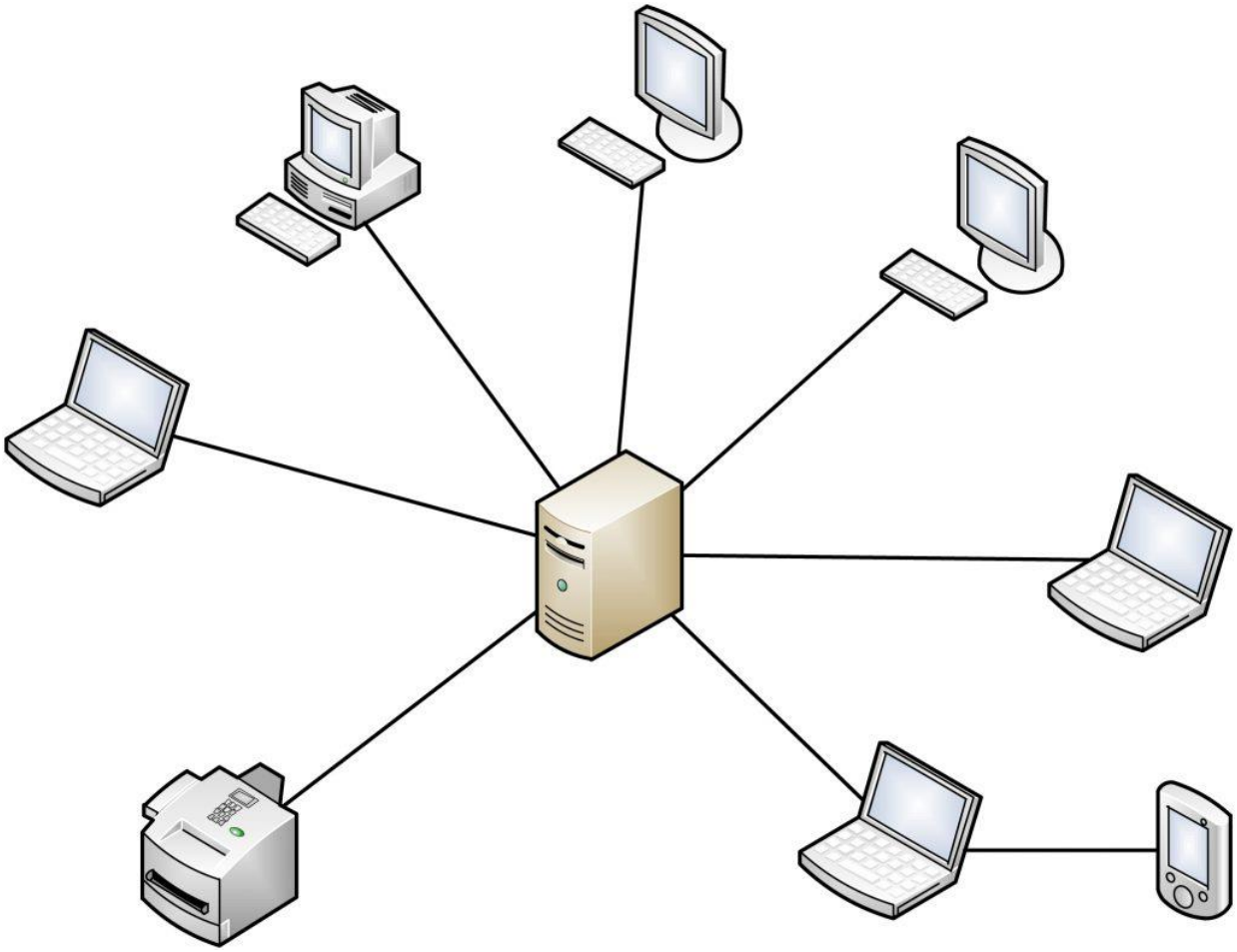
Contents

1.INTRODUCTION

1.1 General Introduction

Client-Server Definition

Client-server denotes a relationship between cooperating programs in an application, composed of clients initiating requests for services and servers providing that function or service.



What is the Client-Server Model?

The client-server model, or client-server architecture, is a distributed application framework dividing tasks between servers and clients, which either reside in the same system or communicate through a computer network or the Internet. The client relies on sending a request to another program in order to access a service made available by a server. The server runs one or more programs that share resources with and distribute work among clients.

The client server relationship communicates in a request–response messaging pattern and must adhere to a common communications protocol, which formally defines the rules, language, and dialog patterns to be used. Client-server communication typically adheres to the TCP/IP protocol suite.

TCP protocol maintains a connection until the client and server have completed the message exchange. TCP protocol determines the best way to distribute application data into packets that networks can deliver, transfers packets to and receives packets from the network, and manages flow control and retransmission of dropped or garbled packets. IP is a connectionless protocol in which each packet traveling through the Internet is an independent unit of data unrelated to any other data units.
Client requests are organized and prioritized in a scheduling system, which helps servers cope in the instance of receiving requests from many distinct clients in a short space of time. The client-server approach enables any general-purpose computer to expand its capabilities by utilizing the shared resources of other hosts. Popular client-server applications include email, the World Wide Web, and network printing.

Categories of Client-Server Computing
There are four main categories of client-server computing:

- One-Tier architecture: consists of a simple program running on a single computer without requiring access to the network. User requests don't manage any network protocols, therefore the code is simple and the network is relieved of the extra traffic.
- Two-Tier architecture: consists of the client, the server, and the protocol that links the two tiers. The Graphical User Interface code resides on the client host and the domain logic resides on the server host. The client-server GUI is written in high-level languages such as C++ and Java.
- Three-Tier architecture: consists of a presentation tier, which is the User Interface layer, the application tier, which is the service layer that performs detailed processing, and the data tier, which consists of a database server that stores information.
- N-Tier architecture: divides an application into logical layers, which separate responsibilities and manage dependencies, and physical tiers, which run on separate machines, improve scalability, and add latency from the additional network communication. N-Tier architecture can be closed-layer, in which a layer can only communicate with the next layer down, or open-layer, in which a layer can communicate with any layers below it.

Microsoft MySQL Server is a popular example of a three-tier architecture, consisting of three major components: a protocol layer, a relational engine, and a storage engine. Any client machines that connect directly to SQL Server must have a SQL Server client installed. Microsoft's Client-Server Runtime Process helps manage the majority of the graphical instruction sets on Windows operating system.

What is a Client-Server Network?
A client-server network is the medium through which clients access resources and services from a central computer, via either a local area network (LAN) or a wide-area network (WAN), such as the Internet. A unique server called a daemon may be employed for the sole purpose of awaiting client requests, at which point the network connection is initiated until the client request has been fulfilled. Network traffic is categorized as client-to-server (north-south traffic) or server-to-server (east-west traffic). Popular network services include e-mail, file sharing, printing, and the World Wide Web. A major advantage of the client-server network is the central management of applications and data.

Benefits of Client-Server Computing
There are numerous advantages of the client server architecture model:

- A single server hosting all the required data in a single place facilitates easy protection of data and management of user authorization and authentication.
- Resources such as network segments, servers, and computers can be added to a client-server network without any significant interruptions.
- Data can be accessed efficiently without requiring clients and the server to be in close proximity.
- All nodes in the client-server system are independent, requesting data only from the server, which facilitates easy upgrades, replacements, and relocation of the nodes.
- Data that is transferred through client-server protocols are platform-agnostic.

Difference Between Client and Server

Clients, also known as service requesters, are pieces of computer hardware or server software that request resources and services made available by a server. Client computing is classified as Thick, Thin, or Hybrid.

- Thick Client: a client that provides rich functionality, performs the majority of data processing itself, and relies very lightly upon the server.
- Thin Client: a thin-client server is a lightweight computer that relies heavily on the resources of the host computer -- an application server performs the majority of any required data processing.
- Hybrid Client: possessing a combination of thin client and thick client characteristics, a hybrid client relies on the server to store persistent data, but is capable of local processing.

A server is a device or computer program that provides functionality for other devices or programs. Any computerized process that can be used or called upon by a client to share resources and distribute work is a server. Some common examples of servers include:

- Application Server: hosts web applications that users in the network can use without needing their own copy.
- Computing Server: shares an enormous amount of computer resources with networked computers that require more CPU power and RAM than is typically available for a personal computer.
- Database Server: maintains and shares databases for any computer program that ingests well-organized data, such as accounting software and spreadsheets.
- Web Server: hosts web pages and facilitates the existence of the World Wide Web.

Difference Between Server-Side Programming and Client-Side Programming

Server-side programming refers to a program that runs on the server and focuses on the generation of dynamic content. Server-side programming is used for querying and interacting with the database, accessing files on a server, interacting with other servers, processing user input, and structuring web applications. Popular programming languages for server-side programming include C++, Java and JSP, PHP, Python, and Ruby on Rails.

Client-side programming refers to a program that runs on the client machine and focuses on the user interface and other processes such as reading and/or writing cookies. Client-side programming is used for sending requests to the server, interacting with local storage, interacting with temporary storage, creating interactive web pages, and functions as an interface between client and server. Popular

programming languages for client-server programming include AJAX, CSS, HTML, Javascript, and VBScript.

Server-Side Rendering vs Client-Side Rendering
Server-side rendering refers to an application's ability to convert HTML files on the server into a fully rendered page for the client. The web browser makes a request for information from the server, which responds, typically in milliseconds, with the fully rendered HTML display. Search engines are able to index and crawl content before it is delivered, making server-side rendering very beneficial for SEO.
In client-server rendering, rather than receiving all of the content from the HTML document, content is rendered in the browser using the client-side JavaScript library. The browser does not make a new request to the server when a new page is loaded. Search engine rankings may be negatively impacted as the content is not rendered until the page is loaded on the browser, however, website rendering tends to be faster in client-side rendering.

Client-Server vs Peer-to-Peer
Peer-to-peer (P2P) is a decentralized communications model in which all nodes in the network have equivalent capability and can function as both a client and server. Nodes in peer-to-peer computing collectively use their resources and communicate with each other directly on-demand.
An algorithm in the peer-to-peer communications protocol balances load, making other peers available to compensate for any resource downtime, and rerouting requests as the load capacity and availability of peers changes. A major advantage of peer-to-peer networking is the ability to expand the network to manage a large number of clients.
In client-server computing, a centralized communications model, the server is the central node that communicates with other client nodes. A major advantage that the client-server relationship has over the peer-to-peer relationship is the ability to manage data and applications in one, centralized server.

Does HEAVY.AI Offer a Client-Server Solution?
HEAVY.AI Render leverages server-side GPUs to instantly render interactive visualizations of high-cardinality data. Exploiting server-side rendering technology, HEAVY.AI can import and display millions of lines of data over the network to the client without any slowdowns associated with the transfer of high cardinality data. This distinguishes HEAVY.AI from other technologies that transfer results to the client for rendering, which slows overall performance.

Client-server Architecture

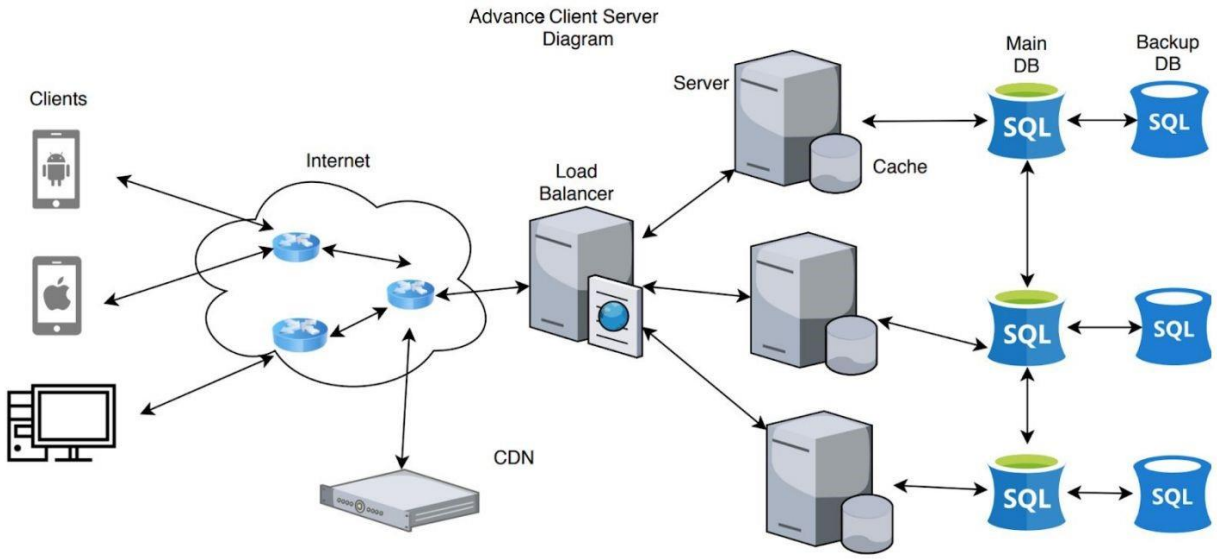
Definition

- Client-server architecture is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client. This type of architecture has one or more client computers connected to a central server over a network or

internet connection. Client-server architecture is also known as a networking computing model or client-server network because all the requests and services are delivered over a network.

- **Vocabulary for components and connectors:**
  - Client: a piece of software or application that takes the input and sends request to the servers.
  - Server: a piece of software that receives and processes requests from clients.
    - Load balancer: responsible for distributing incoming network traffic across a group of backend servers to optimize resource usage
  - Network layer protocols such as TCP/IP

Topological constraints



The flow of the data is unidirectional and forms a cycle. It is usually initiated by the client requesting some kind of data and the server processing the request and sending some kind of data back to the client via a protocol. Clients cannot directly talk to each other.

A typical topological data flow goes as follows:

1. Client requests data from server
2. Load balancer routes the request to the appropriate server
3. Server processes the request client
4. Server queries appropriate database for some data
5. Database returns the queried data back to the server

6. The server processes the data and sends the data back to the client
7. This process repeats

Applicable problems

- The client-server architecture is most useful for applications that require a separation or abstraction of concerns between the client and the server; it is meant for systems with high interoperability. The client-server architectural style helps applications improve performance in scalability.
- In systems that need separation of functionality, the client-server architecture design is most applicable. Request validation and input could be handled from the client side while the load balancer routes the request to the server for adequate processing. The server will be responsible for processing the client's request, and returning the result via the right protocol. These layers (client and server) complete tasks independently and they are useful for abstracting functionality; for example, the client does not need to know how the server handles user authentication or request validation.
- With the separation of functionality comes the ability of each layer to function more efficiently at large scale. Modern techniques have been developed within the client-server architecture to solve scalability challenges like load balancing, sharding, and partitioning. These techniques provide performance improvements for multiple requests on the server side of the architecture and will be useful for software programs that deal with multiple requests/users.

Resilience to change

- This architecture style is extremely flexible and can be adapted to the user and problem set.
- It can also be combined with other architecture types on the client or server sides.
- As functional and nonfunctional requirements change, modules can be updated without altering the client-server architecture or disrupting service.
- Since the data being passed between the client and server, and services the server provides are entirely up to the developers there is an infinite amount of ways to use this architecture style to solve problems that may arise in the future.

Negative behaviours

- Could be a greater potential for failure because of centralized control: ○ Servers help in administering the whole set-up
    - If the servers go down then entire service goes down
- Particularly vulnerable to Denial of Service (DOS) attacks because the number of servers is considerably smaller than the number of clients
    - DOS attack is a cyber-attack in which perpetrators aim to make a network resource unavailable
        - Done by flooding the resource with requests in an attempt to overload the system
        - Results in legitimate requests not being fulfilled
    - DDOS attack is a distributed denial-of-service attack
        - The incoming traffic comes from multiple clients
        - Cannot stop the attack by blocking a single client
- Very expensive to install and manage the network
    - Server machines are much more powerful than standard client machines, which means that they are more expensive
    - Requires hiring employees with networking and infrastructure knowledge, in order to manage the system

Supported NFPs

- Scalability: capable of horizontal or vertical scaling of the system
    - Horizontal scaling: adding or removing servers in the network
    - Vertical scaling: migrating to larger and faster server machines
- Availability: servers typically do not have to shutdown or restart for many days
    - Server uptime is possible during maintenance, with server duplication
- Dependability: processing and resource allocation is done by a dedicated set of machines
    - These servers can be optimized to complete a certain task quickly and efficiently
- Heterogeneity: clients are consumers of services and servers are providers of services
    - Clear separation of concerns results in the system being composed of disparate parts

- For example, one or more servers may fail, but the system can still function as long as the other servers offer the same services

Inhibited NFPs

- Robustness: if many clients send simultaneous requests to the server, and there is only a single centralized server, then this might overload the server and slow down the performance drastically. This could possibly lead to a server failure, in which case the whole system goes down and the clients are not able to get any responses back.
- Transparency: since clients only make requests to the server with their input data, they don't see how the data will be distributed across the servers. It may seem like there only exists a single, central server for a user.

Comparison with other architectures

- Client-server, layered, and pipe and filter architectures are similar in their objective.
  - Client-server can be thought of as a variation of layered architecture with two layers.
  - Pipe and filter only allows unidirectional flow of information, whereas client-server and layered architectures allow bidirectional flow.

Client-Server Model

- Difficulty Level : Basic
- Last Updated : 15 Nov, 2019

The Client-server model is a distributed application structure that partitions task or workload between the providers of a resource or service, called servers, and service requesters called clients. In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and deliver the data packets requested back to the client. Clients do not share any of their resources. Examples of Client-Server Model are Email, World Wide Web, etc.

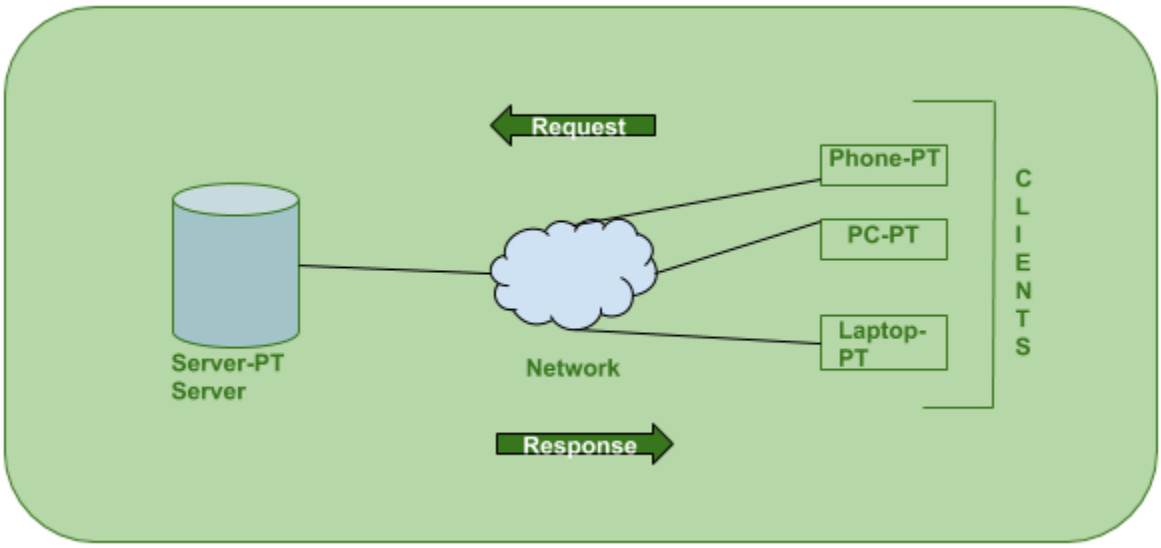
How the Client-Server Model works ?

In this article we are going to take a dive into the **Client-Server** model and have a look at how the **Internet** works via, web browsers. This article will help us in

having a solid foundation of the WEB and help in working with WEB technologies with ease.

- **Client:** When we talk the word **Client**, it mean to talk of a person or an organization using a particular service. Similarly in the digital world a **Client** is a computer (**Host**) i.e. capable of receiving information or using a particular service from the service providers (**Servers**).
- **Servers:** Similarly, when we talk the word **Servers**, It mean a person or medium that serves something. Similarly in this digital world a **Server** is a remote computer which provides information (data) or access to particular services.

So, its basically the **Client** requesting something and the **Server** serving it as long as its present in the database.

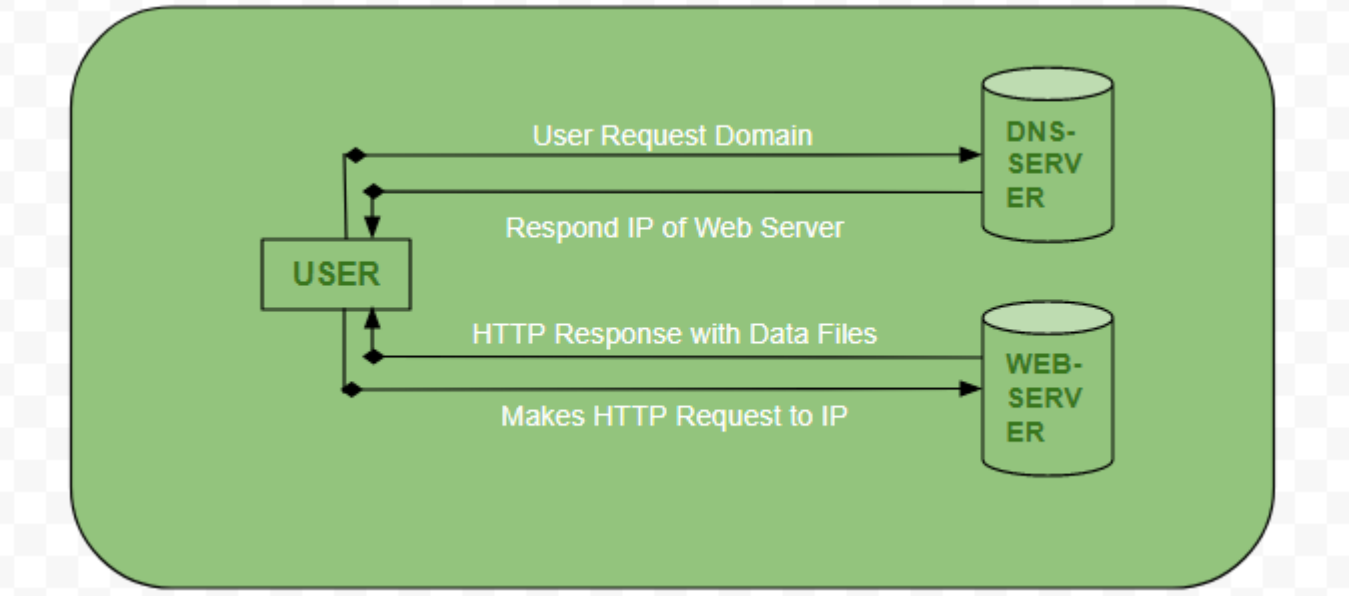


How the browser interacts with the servers ?

There are few steps to follow to interacts with the servers a client.

- User enters the **URL**(Uniform Resource Locator) of the website or file. The Browser then requests the **DNS**(DOMAIN NAME SYSTEM) Server.
- **DNS Server** lookup for the address of the **WEB Server**.
- **DNS Server** responds with the **IP address** of the **WEB Server**.
- Browser sends over an **HTTP/HTTPS** request to **WEB Server's IP** (provided by **DNS server**).
- Server sends over the necessary files of the website.
- Browser then renders the files and the website is displayed. This rendering is done with the help of **DOM** (Document Object Model)

interpreter, **CSS** interpreter and **JS Engine** collectively known as the **JIT** or (Just in Time) Compilers.



Advantages of Client-Server model:

- Centralized system with all data in a single place.
- Cost efficient requires less maintenance cost and Data recovery is possible.
- The capacity of the Client and Servers can be changed separately.

Disadvantages of Client-Server model:

- Clients are prone to viruses, Trojans and worms if present in the Server or uploaded into the Server.
- Server are prone to Denial of Service (DOS) attacks.
- Data packets may be spoofed or modified during transmission.
- Phishing or capturing login credentials or other useful information of the user are common and MITM(Man in the Middle) attacks are common.

Client and server communication[edit]

Generally, a service is an abstraction of computer resources and a client does not have to be concerned with how the server performs while fulfilling the request and delivering the response. The client only has to understand the response based on the well-known application protocol, i.e. the content and the formatting of the data for the requested service.

Clients and servers exchange messages in a request–response messaging pattern. The client sends a request, and the server returns a response. This exchange of messages is an example of inter-process communication. To communicate, the computers must have a common language, and they must follow rules so that both the client and the server know what to expect. The language and rules of communication are defined in a communications protocol. All protocols operate in the application layer. The application layer protocol defines the basic patterns of the dialogue. To formalize the data exchange even further, the server may implement an application programming interface (API).[3] The

API is an abstraction layer for accessing a service. By restricting communication to a specific content format, it facilitates parsing. By abstracting access, it facilitates cross-platform data exchange.[4]

A server may receive requests from many distinct clients in a short period. A computer can only perform a limited number of tasks at any moment, and relies on a scheduling system to prioritize incoming requests from clients to accommodate them. To prevent abuse and maximize availability, the server software may limit the availability to clients. Denial of service attacks are designed to exploit a server's obligation to process requests by overloading it with excessive request rates. Encryption should be applied if sensitive information is to be communicated between the client and the server.

Client-host and server-host[edit]

Client-host and *server-host* have subtly different meanings than *client* and *server*. A host is any computer connected to a network. Whereas the words *server* and *client* may refer either to a computer or to a computer program, *server-host* and *client-host* always refer to computers. The host is a versatile, multifunction computer; *clients* and *servers* are just programs that run on a host. In the client-server model, a server is more likely to be devoted to the task of serving.

An early use of the word *client* occurs in "Separating Data from Function in a Distributed File System", a 1978 paper by Xerox PARC computer scientists Howard Sturgis, James Mitchell, and Jay Israel. The authors are careful to define the term for readers, and explain that they use it to distinguish between the user and the user's network node (the client).[7] By 1992, the word *server* had entered into general parlance.[8][9]

Centralized computing[edit]

Further information: History of personal computers, Decentralized computing, and Computer cluster

The client-server model does not dictate that server-hosts must have more resources than client-hosts. Rather, it enables any general-purpose computer to extend its capabilities by using the shared resources of other hosts. Centralized computing, however, specifically allocates a large number of resources to a small number of computers. The more computation is offloaded from client-hosts to the central computers, the simpler the client-hosts can be.[10] It relies heavily on network resources (servers and infrastructure) for computation and storage. A diskless node loads even its operating system from the network, and a computer terminal has no operating system at all; it is only an input/output interface to the server. In contrast, a rich client, such as a personal computer, has many resources and does not rely on a server for essential functions.

As microcomputers decreased in price and increased in power from the 1980s to the late 1990s, many organizations transitioned computation from centralized servers, such as mainframes and minicomputers, to rich clients.[11] This afforded greater, more individualized dominion over computer resources, but complicated information technology management.[10][12][13] During the 2000s, web applications matured enough to rival application software developed for a specific microarchitecture. This maturation, more affordable mass storage, and the advent of service-oriented architecture were among the factors that gave rise to the cloud computing trend of the 2010s.[14]

Comparison with peer-to-peer architecture[edit]

In addition to the client-server model, distributed computing applications often use the peer-to-peer (P2P) application architecture.

In the client–server model, the server is often designed to operate as a centralized system that serves many clients. The computing power, memory and storage requirements of a server must be

scaled appropriately to the expected workload. Load-balancing and failover systems are often employed to scale the server beyond a single physical machine.[15][16]

Load balancing is defined as the methodical and efficient distribution of network or application traffic across multiple servers in a server farm. Each load balancer sits between client devices and backend servers, receiving and then distributing incoming requests to any available server capable of fulfilling them.

In a peer-to-peer network, two or more computers (*peers*) pool their resources and communicate in a decentralized system. Peers are coequal, or equipotent nodes in a non-hierarchical network. Unlike clients in a client-server or client–queue–client network, peers communicate with each other directly.[citation needed] In peer-to-peer networking, an algorithm in the peer-to-peer communications protocol balances load, and even peers with modest resources can help to share the load.[citation needed] If a node becomes unavailable, its shared resources remain available as long as other peers offer it. Ideally, a peer does not need to achieve high availability because other, redundant peers make up for any resource downtime; as the availability and load capacity of peers change, the protocol reroutes requests.

Both client–server and master–slave are regarded as sub-categories of distributed peer-to-peer systems.

Client/ Server technology is a means for separating the functions of an application into two or more distinct parts. Client/ server describes the relationship between two computer programs in which one program, the client, makes a service request from another program, the server, which fulfills the request. The client presents and manipulates data on the desktop computer. The server acts like a mainframe to store and retrieve protected data. It is network architecture in which each computer or process on the network is either a client or a server. Servers are powerful computers or processes dedicated to managing disk drives (file servers), printers (print servers), or network traffic (network servers). Clients are PCs or workstations on which users run applications. Clients rely on servers for resources, such as files, devices, and even processing power.
In this type of architecture, storage devices are connected to a server, and users get connected to this server to access the data on the storage devices. If this server goes down, the data on the storage can't be retrieved. Adding one more server increases the probability of availability of data, but that is to a very minimal extent. Contrast to the cluster case, here though there are two servers, only one server can access the Storage devices at any point in time.

Servers and Data storage are connected using SCSI cables. Important limitations in this architecture here are:

1. Number of Storage devices connected to one server here is limited, because the each server can accommodate limited number of SCSI cards.
2. Length of the SCSI cable is limited to 25 meters.
3. The scalability which is much more essential and needed in terms of storage capacity here, is limited. For example: As per the recent posts by EMC and research firm IDC, 487 billion giga bytes of digital data is created in 2008. They say, this

value is getting doubled every 18 months, so when the data increase is going up so rapidly, Server-Centric Architecture is not going to work.

4. The unused storage connected to one server, can't be used by the second server, though the later ran out of space completely.
5. **Client and Server Hardware**

6. Client/server networking grew in popularity many years ago as personal computers (PCs) became the common alternative to older mainframe computers.

7. Client devices are typically PCs with network software applications installed that request and receive information over the network. Mobile devices, as well as desktop computers, can both function as clients.

8. A server device typically stores files and databases including more complex applications like Web sites. Server devices often feature higher-powered central processors, more memory, and larger disk drives than clients.

9. **Client-Server Applications**

10. The client-server model organizes network traffic by a client application and also by a device. Network clients send messages to a server to make requests of it. Servers respond to their clients by acting on each request and returning results. One server supports many clients, and multiple servers can be networked together in a server pool to handle increased processing loads as the number of clients grows.

11. A client computer and a server computer are usually two separate units of hardware each customized for their designed purpose.

12. For example, a Web client works best with a large screen display, while a Web server does not need any display at all and can be located anywhere in the world. In some cases, however, a given device can function both as a client and a server for the same application. Additionally, a device that is a server for one application can simultaneously act as a client to other servers, for different applications.

13. Some of the most popular applications on the Internet follow the client-server model including email, FTP and Web services. Each of these clients features a user interface (either graphic or text-based) and a client application that allows the user to connect to servers. In the case of email and FTP, users enter a computer name (or sometimes an IP address) into the interface to set up connections to the server.

14. **Local Client-Server Networks**

15. Many home networks utilize client-server systems on a small scale. Broadband routers, for example, contain DHCP servers that provide IP addresses to the home

computers (DHCP clients). Other types of network servers found in home include print servers and backup servers.

16. **Client-Server vs. Peer-to-Peer and Other Models**

17. The client-server model of networking was originally developed to share access to database applications among larger numbers of users. Compared to the mainframe model, client-server networking gives better flexibility as connections can be made on-demand as needed rather than being fixed. The client-server model also supports modular applications that can make the job of creating software easier. In so-called two tier and three tier types of client-server systems, software applications are separated into modular components, and each component is installed on clients or servers specialized for that subsystem.

18. Client-server is just one approach to managing network applications. The primary alternative to client-server, peer-to-peer networking, treats all devices as having equivalent capability rather than specialized client or server roles. Compared to client-server, peer to peer networks offer some advantages such as better flexibility in expanding the network to handle a large number of clients. Client-server networks generally offer advantages over peer-to-peer as well, such as the ability to manage applications and data in one centralized location.

What is client server architecture?

The client-server architecture is also termed as a network-computing structure because every request and their associated services are distributed over a network. So now the question is how the thing works? In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested, process it and deliver the data packets requested back to the client. One special feature is that the server computer has the potential to manage numerous clients at the same time. Also, a single client can connect to numerous servers at a single timestamp, where each server provides a different set of services to that specific client.

What are two examples of server side and client?

You can think of the ‘Server’ as an object that provides data, and the ‘Client’ as an object that receives and processes this data for the end user.

This is a very crude analogy, but imagine a restaurant; the diner is the end user. In order for the diner to eat, the chef (client) must prepare a meal using various ingredients. **But where do these ingredients come from?**

If the meal is an omelette, the core ingredient is an egg. Which is supplied by a chicken, which in turn is raised and looked after by a **farmer**. The farmer is the server; he rears the chickens, gets rid of their poo, feeds them, and when they lay their eggs, the farmer processes them (sometimes even stamping **dates** to indicate how long they are good for, and **where they have come from**, and other information about the egg a.k.a **meta-data)**.

Once the **farmer** has the eggs prepared and processed, ready for the **chef** (client), to use, he packages them so that they won't break and in a secure vehicle to avoid theft. He then transports them to the **chef** (client), so that the meal can be prepared for the **diner** (end-user).

As you can see from this quick little analogy, the client and server must work in tandem in order to deliver the result to the end-user, which in this case is the **diner**. But there's a very apparent difference between the client (chef) and the server (farmer).

The **chef** is within the immediate vicinity of the **diner**; he has entered the **chef's** restaurant, and has made a request to the **chef** personally. If he so wished, he could walk to the back of the kitchen and **ask the chef where the ingredients were coming from**.

The **farmer**, on the other hand, is not in the vicinity of the **diner**. He cannot be interrogated by the **diner**, and in some cases may be hundreds or thousands of miles away. His only task is to provide the **ingredients** that the **chef** has requested.

Let's look at another example, this time in the context of the internet. You have a web-browser (chrome ftw). This is the **client**, and you are the **end-user**. You wish to ask a question on Quora, so you write down http://www.quora.com in the address bar and click enter. What happens next?

Well first, your **client** makes a request to Quora's **server**. It says 'Hi Quora **Server**, I want to see this page: http://www.quora.com.'

This request is received by the **server,** which then prepares all the data needed for the **client** to show you that page. What does the server do to prepare the data? Well, maybe it needs to run some code that is on the server to generate a particular section of the page. Or it needs to query the database in order to pull the latest questions and answers. In any case, the **server** prepares all these things, puts them in a format that the **client** can understand (i.e HTML, Javascript, and CSS), and sends them to the client. This is what we call the **response**. Everything that occurs before the **response** can be created and sent back to the **client** is what is known as **server-side**.

NB: This is a very crude analogy that will likely break under great scrutiny. In hindsight, the **client** should probably be the **restaurant** rather then the **chef.**

Hope this makes it easier to understand! Any improvements on this answer would be greatly appreciated. :)

What is a client/server system with an example?

Traditionally, the client is the application running on the computer you are actually using and the server is the application running on a remote computer that is actually doing something for you.

So when you are doing a Google search, you type in your search, into your local browser and this client browser sends the search details, over the Internet, to one of Googles search server applications, that application then runs the search over it's database and returns the results back to the client browser which displays the results on your computer.

Technically, the server does not have to be on a remote computer, so particularly while developing a server application it is normal to have the development machine running both the client and server applications at the same time. The way the two applications talk to each other is the same as when they are on different machines, the messages just don't have to travel as far!

What is the difference between server and client?

Originally Answered: What are the differences between clients and servers?

In computing terminology, both "client" and "server" refer to computers that are used for different purposes.

A client is a small computer that accesses a server through a network. For example, in an organization, an employee logs in to the client machine to access the files and applications running on a server machine.

A server machine is a large-capacity computer that can store a wide variety of files such as application and data files. There are various types of servers, such as an application server, file server, web server, database server, print server, proxy server, game server, standalone server

The main difference between a client machine and a server machine is in its performance. The client machines are considered optimal for applications which require speedy start-up times. A server machine is considered optimal for applications where the emphasis is more on performance.

Summary:

1. A client machine is a small computer with a basic hardware configuration whereas

a server machine is a high-end computer with an advanced hardware configuration.

2. A client is a simple and less powerful machine whereas a server is a powerful expensive machine.

3. A client is used for simple tasks whereas a server is used for storing huge data files and applications.

4. A server delivers high performance compared to a client machine.

5. A server supports simultaneous, multiple user log-ins whereas a client supports a single user log-in at a time.