

Name : Raha Fuad Habib  
Reg No. COM/0501/21  
Unit : Client Server Computing  
Course Code: CSC226  
Kibabii University

**Client Server Computing Models**

The foundation of modern networking is the centralized process and IT framework known as the client-server model. For over 50 years, [servers](https://www.serverwatch.com/guides/what-is-a-server/) have been the machines and mechanisms to process end-user requests and deliver specific digital resources. These requests include anything from a Google search on a smartphone to a remote employee accessing confidential company information.  
In both instances, a user device triggers a request to another machine, like a [network server](https://www.serverwatch.com/servers/network-server/), which receives, filters, processes, redirects, or any combination of these actions before returning the adequate response.

The client-server model is a network architecture that describes how servers share resources and interact with network devices. For modern enterprises and [data centers](https://www.serverwatch.com/servers/colocation-providers/), many servers facilitate processes like email, [printing](https://www.serverwatch.com/guides/print-server/), internet connections, [application hosting](https://www.serverwatch.com/guides/application-server/), and more.

The client-server model describes how network devices like workstations, laptops, and IoT devices — known as clients — make requests to network machines or software capable of completing the request, known as servers.  
Though servers historically have been physical appliances like rack servers, data center trends show administrators increasingly deploy virtual servers for a range of workloads.  
Enterprises once relied on workstations that shared a single operating system, but today’s IT environment strives for interoperability between systems, devices, applications, and products.  
Because network clients are increasingly heterogeneous device and OS types, administrators have the added responsibility of ensuring compatibility.  
The hyper-focused machines delivering IT resources to clients are the physical, virtual, and [cloud-based](https://www.serverwatch.com/cloud/cloud-based-services/) servers housed in modern data centers. Servers are the central authority for several essential digital processes that enterprise organizations rely on.

A client-server web application is a software architecture where tasks are divided between a client (user interface) and a server (backend). The client sends requests to the server, which processes them and returns the results, enabling dynamic and interactive web experiences.  Users interact with a client-side interface, sending requests to a server. The server processes these requests, performs necessary tasks, and sends back the results to the client. This architecture allows for efficient division of responsibilities between the user interface and backend processing.

Client-server web applications operate based on a robust communication model in which the client initiates requests to the server, and subsequently, the server meticulously processes these requests before furnishing the essential data back to the client. This well-structured architecture not only facilitates swift and effective data retrieval, processing, and presentation on the user interface but also ensures a seamless user experience. The significance of the body of client-server web applications is paramount as it intricately governs the transmission, processing, and presentation of data to users, thereby shaping the overall functionality and usability of the application.

## [Web servers and HTTP (a primer)](https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview#web_servers_and_http_a_primer)

Web browsers communicate with [web servers](https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_web_server) using the **H**yper**T**ext **T**ransfer **P**rotocol ([HTTP](https://developer.mozilla.org/en-US/docs/Web/HTTP)). When you click a link on a web page, submit a form, or run a search, the browser sends an HTTP Request to the server.

This request includes:

* A URL identifying the target server and resource (e.g. an HTML file, a particular data point on the server, or a tool to run).
* A method that defines the required action (for example, to get a file or to save or update some data). The different methods/verbs and their associated actions are listed below:
  + GET: Get a specific resource (e.g. an HTML file containing information about a product, or a list of products).
  + POST: Create a new resource (e.g. add a new article to a wiki, add a new contact to a database).
  + HEAD: Get the metadata information about a specific resource without getting the body like GET would. You might for example use a HEAD request to find out the last time a resource was updated, and then only use the (more "expensive") GET request to download the resource if it has changed.
  + PUT: Update an existing resource (or create a new one if it doesn't exist).
  + DELETE: Delete the specified resource.
  + TRACE, OPTIONS, CONNECT, PATCH: These verbs are for less common/advanced tasks, so we won't cover them here.
* Additional information can be encoded with the request (for example, HTML form data). Information can be encoded as:
  + URL parameters: GET requests encode data in the URL sent to the server by adding name/value pairs onto the end of it — for example http://example.com?name=Fred&age=11. You always have a question mark (?) separating the rest of the URL from the URL parameters, an equals sign (=) separating each name from its associated value, and an ampersand (&) separating each pair. URL parameters are inherently "insecure" as they can be changed by users and then resubmitted. As a result URL parameters/GET requests are not used for requests that update data on the server.
  + POST data. POST requests add new resources, the data for which is encoded within the request body.
  + Client-side cookies. Cookies contain session data about the client, including keys that the server can use to determine their login status and permissions/accesses to resources.

Web servers wait for client request messages, process them when they arrive, and reply to the web browser with an HTTP Response message. The response contains an [HTTP Response status code](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status) indicating whether or not the request succeeded (e.g. "200 OK" for success, "404 Not Found" if the resource cannot be found, "403 Forbidden" if the user isn't authorized to see the resource, etc.). The body of a successful response to a GET request would contain the requested resource.

When an HTML page is returned it is rendered by the web browser. As part of processing, the browser may discover links to other resources (e.g. an HTML page usually references JavaScript and CSS files), and will send separate HTTP Requests to download these files.

Both static and dynamic websites (discussed in the following sections) use exactly the same communication protocol/patterns.

You can make a simple GET request by clicking on a link or searching on a site (like a search engine homepage). For example, the HTTP request that is sent when you perform a search on MDN for the term "client-server overview" will look a lot like the text shown below (it will not be identical because parts of the message depend on your browser/setup).

The Different Types of Client-Server Architectures

The four types of client-server frameworks are representative of how the client-server relationship evolved with networking advancements.

**1-Tier: All-in-One System**

All client-server configuration settings, user interface, business logic, and database logic sit on network devices in the base client-server architecture. Often limited to smaller networks, 1-Tier frameworks include presentation, business, and data access layers on the same machine.

**2-Tier: Client and Server**

2-Tier architectures add a server to the mix and separate the presentation layer to a user interface, where the client makes requests outside its capabilities. Clients and servers take on more business and database logic at varying levels, providing administrators some control.

**3-Tier: Middleware**

To further protect and enrich the 2-Tier architecture, this framework incorporates middleware between the client tier (presentation layer) and the server tier (database layer).

This application layer provides a third tier, enabling more complex management of business logic. Examples of middleware like web application servers offer load balancing, increased storage, and security.

**N-Tier: Multitier Architecture**

Working off of the 3-Tier architecture, N-Tier describes the further use of middleware to segment network traffic and functions. Though this comes with greater complexity, modern enterprise organizations require multitier architectures’ flexibility, scalability, and security.

Peer-to-Peer vs Client-Server

Though the client-server framework is the most popular distributed architecture, the most common alternative mentioned is peer-to-peer (P2P) networks.

Unlike the client-server architecture, which offers a centralized flow from an organization’s servers to staff, stakeholders, and clients, a peer-to-peer network is decentralized. Network devices don’t align with a specific, dedicated server. Instead, P2P networks include network devices (clients) known as nodes that submit and service requests between each other.

Because P2P networks like [blockchains](https://www.esecurityplanet.com/applications/cybersecurity-blockchain-applications/" \t "_blank) work in a distributed fashion, connectivity is critical to operations. Relative to traditional networks, nodes tend to be more heterogeneous and share less data across the network.

The client-server model is more expensive to set up and maintain, but the reliability and scalability of the centralized framework consistently outweigh the downfalls of peer-to-peer frameworks.

The Distributed Architecture of Choice for Enterprises

With benefits like centralized data control, [workload balancing](https://www.serverwatch.com/servers/load-balancing/), added security, and redundancy, the client-server model is the near-universal framework for SMBs up to enterprise organizations.

When compared to P2P networks, it’s easy to see why the client-server model scaled digital operations for networks to this point. As organizations grow, adopting additional tiers of client-server architecture is necessary to maintain IT infrastructure and secure data integrity properly.

To serve millions of customers and stakeholders globally, enterprises need a robust distributed framework that only the client-server model provides.

Client-server web applications operate based on a robust communication model in which the client initiates requests to the server, and subsequently, the server meticulously processes these requests before furnishing the essential data back to the client. This well-structured architecture not only facilitates swift and effective data retrieval, processing, and presentation on the user interface but also ensures a seamless user experience. The significance of the body of client-server web applications is paramount as it intricately governs the transmission, processing, and presentation of data to users, thereby shaping the overall functionality and usability of the application.

In conclusion, client-server computing models form the foundation of modern technology infrastructure, providing a framework for distributing tasks and resources efficiently. In web applications, this architecture enables seamless interaction between clients and servers, facilitating the development of dynamic and interactive user experiences. By embracing the principles of client-server computing, organizations can build robust and scalable systems that meet the evolving demands of today’s digital landscape.