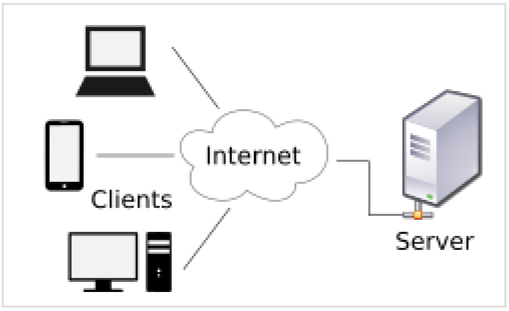
WIKIPEDIA
The Free Encyclopedia

Client–server model

The **client–server model**, also known as client server network architecture, is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.[1] Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs, which share their resources with clients. A client usually does not share any of its resources, but it requests content or service from a server. Clients, therefore, initiate communication sessions with servers, which await incoming requests. Examples of computer applications that use the client–server model are email, network printing, and the World Wide Web.



A computer network diagram of clients communicating with a server via the Internet

Client and server role

The "client-server" characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services. Servers are classified by the services they provide. For example, a web server serves web pages and a file server serves computer files. A shared resource may be any of the server computer's software and electronic components, from programs and data to processors and storage devices. The sharing of resources of a server constitutes a *service*.

Whether a computer is a client, a server, or both, is determined by the nature of the application that requires the service functions. For example, a single computer can run a web server and file server software at the same time to serve different data to clients making different kinds of requests. The client software can also communicate with server software within the same computer.[2] Communication between servers, such as to synchronize data, is sometimes called *inter-server* or *server-to-server* communication.

Client and server communication

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.

Example

When a bank customer accesses online banking services with a web browser (the client), the client initiates a request to the bank's web server. The customer's login credentials may be s

Server-side

Server-side refers to programs and operations that run on the server. This is in contrast to client-side programs and operations which run on the client.[5] (See below)

General concepts

"Server-side software" refers to a computer application, such as a web server, that runs on remote server hardware, reachable from a user's local computer, smartphone, or other device. Operations may be performed server-side because they require access to information or functionality that is not available on the client, or because performing such operations on the client side would be slow, unreliable, or insecure.

Client and server programs may be commonly available ones such as free or commercial web servers and web browsers, communicating with each other using standardized protocols. Or, programmers may write their own server, client, and communications protocol which can only be used with one another.

Server-side operations include both those that are carried out in response to client requests, and non-client-oriented operations such as maintenance tasks.[6][7]

Computer security

In a computer security context, server-side vulnerabilities or attacks refer to those that occur on a server computer system, rather than on the client side, or in between the two. For example, an attacker might exploit an SQL injection vulnerability in a web application in order to maliciously change or gain unauthorized access to data in the server's database. Alternatively, an attacker might break into a server system using vulnerabilities in the underlying operating system and then be able to access database and other files in the same manner as authorized administrators of the server.[8][9][10]

Examples

In the case of distributed computing projects such as SETI@home and the Great Internet Mersenne Prime Search, while the bulk of the operations occur on the client side, the servers are responsible for coordinating the clients, sending them data to analyze, receiving and storing results, providing reporting functionality to project administrators, etc. In the case of an Internet-dependent user application like Google Earth, while querying and display of map data takes place on the client side, the server is responsible for permanent storage of map data, resolving user queries into map data to be returned to the client, etc.

In the context of the World Wide Web, commonly encountered server-side computer languages include:[5]

- C# or Visual Basic in ASP.NET environments
- Java
- Perl
- PHP
- Python
- Ruby
- Node.js
- Swift

However, web applications and services can be implemented in almost any language, as long as they can return data to standards-based web browsers (possibly via intermediary programs) in formats which they can use.

Client side

Client-side refers to operations that are performed by the client in a computer network.

General concepts

Typically, a client is a computer application, such as a web browser, that runs on a user's local computer, smartphone, or other device, and connects to a server as necessary. Operations may be performed client-side because they require access to information or functionality that is available on the client but not on the server, because the user needs to observe the operations or provide input, or because the server lacks the processing power to perform the operations in a timely manner for all of the clients it serves. Additionally, if operations can be performed by the client, without sending data over the network, they may take less time, use less bandwidth, and incur a lesser security risk.

When the server serves data in a commonly used manner, for example according to standard protocols such as HTTP or FTP, users may have their choice of a number of client programs (e.g. most modern web browsers can request and receive data using both HTTP and FTP). In the case of more specialized applications, programmers may write their own server, client, and communications protocol which can only be used with one another.

Programs that run on a user's local computer without ever sending or receiving data over a network are not considered clients, and so the operations of such programs would not be termed client-side operations.

Computer security

In a computer security context, client-side vulnerabilities or attacks refer to those that occur on the client / user's computer system, rather than on the server side, or in between the two. As an example, if a server contained an encrypted file or message which could only be decrypted using a key housed on the user's computer system, a client-side attack would normally be an attacker's only opportunity to gain access to the decrypted contents. For instance, the attacker might cause malware to be installed on the client system, allowing the attacker to view the user's screen, record the user's keystrokes, and steal copies of the user's encryption keys, etc. Alternatively, an attacker might employ cross-site scripting vulnerabilities to execute malicious code on the client's system without needing to install any permanently resident malware.[8][9][10]

Examples

Distributed computing projects such as SETI@home and the Great Internet Mersenne Prime Search, as well as Internet-dependent applications like Google Earth, rely primarily on client-side operations. They initiate a connection with the server (either in response to a user query, as with Google Earth, or in an automated fashion, as with SETI@home), and request some data. The server selects a data set (a server-side operation) and sends it back to the client. The client then analyzes the data (a client-side operation), and, when the analysis is complete, displays it to the user (as with Google Earth) and/or transmits the results of calculations back to the server (as with SETI@home).

In the context of the World Wide Web, commonly encountered computer languages which are evaluated or run on the client side include:[5]

- Cascading Style Sheets (CSS)
- HTML
- JavaScript

Early history

An early form of client–server architecture is remote job entry, dating at least to OS/360 (announced 1964), where the request was to run a job, and the response was the output.

While formulating the client–server model in the 1960s and 1970s, computer scientists building ARPANET (at the Stanford Research Institute) used the terms *server-host* (or *serving host*) and *user-host* (or *using-host*), and these appear in the early documents RFC 5[11] and RFC 4.[12] This usage was continued at Xerox PARC in the mid-1970s.

One context in which researchers used these terms was in the design of a computer network programming language called Decode-Encode Language (DEL).[11] The purpose of this language was to accept commands from one computer (the user-host), which would return status reports to the user as it encoded the commands in network packets. Another DEL-capable computer, the server-host, received the packets, decoded them, and returned formatted data to the user-host. A DEL program on the user-host received the results to present to the user. This is a client–server transaction. Development of DEL was just beginning in 1969, the year that the United States Department of Defense established ARPANET (predecessor of Internet).

Client-host and server-host

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).[13] By 1992, the word *server* had entered into general parlance.[14][15]

Centralized computing

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.[16] 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.[17] This afforded greater, more individualized dominion over computer resources, but complicated information technology management.[16][18][19] 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.[20]

Comparison with peer-to-peer architecture

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.[21][22]

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. 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. 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.[23]

See also

- Endpoint security
- Front and back ends
- Modular programming
- Observer pattern
- Publish–subscribe pattern
- Pull technology
- Push technology
- Remote procedure call
- Server change number
- Systems Network Architecture, a proprietary network architecture by IBM
- Thin client
- Configurable Network Computing, a proprietary client-server architecture by JD Edwards

Notes

1. "Distributed Application Architecture" (https://web.archive.org/web/20110406121920/http://java.sun.com/developer/Books/jdbc/ch07.pdf) (PDF). Sun Microsystem. Archived from the original (http://java.sun.com/developer/Books/jdbc/ch07.pdf) (PDF) on 6 April 2011. Retrieved 2009-06-16.
2. The X Window System is one example.
3. Benatallah, B.; Casati, F.; Toumani, F. (2004). "Web service conversation modeling: A cornerstone for e-business automation". *IEEE Internet Computing*. **8**: 46–54. doi:10.1109/MIC.2004.1260703 (https://doi.org/10.1109%2FMIC.2004.1260703). S2CID 8121624 (https://api.semanticscholar.org/CorpusID:8121624).
4. Dustdar, S.; Schreiner, W. (2005). "A survey on web services composition" (http://www.infosys.tuwien.ac.at/Staff/sd/papers/A%20survey%20on%20web%20services%20composition_Dustdar_Schreiner_inPress.pdf) (PDF). *International Journal of Web and Grid Services*. **1**: 1. CiteSeerX 10.1.1.139.4827 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.139.4827). doi:10.1504/IJWGS.2005.007545 (https://doi.org/10.1504%2FIJWGS.2005.007545).
5. "What are the differences between server-side and client-side programming?" (http://softwareengineering.stackexchange.com/questions/171203/what-are-the-differences-between-server-side-and-client-side-programming). *softwareengineering.stackexchange.com*. Retrieved 2016-12-13.
6. "Introduction to the server side - Learn web development | MDN" (https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction). *developer.mozilla.org*. 2023-11-05. Retrieved 2023-11-13.
7. "Server-side website programming - Learn web development | MDN" (https://developer.mozilla.org/en-US/docs/Learn/Server-side). *developer.mozilla.org*. 2023-06-30. Retrieved 2023-11-13.

8. Lehtinen, Rick; Russell, Deborah; Gangemi, G. T. (2006). *Computer Security Basics* (https://books.google.com/books?id=DyrLV0kZEd8C&q=client-side+OR+server-side&pg=PT17) (2nd ed.). O'Reilly Media. ISBN 9780596006693. Retrieved 2017-07-07.
9. JS (2015-10-15). "Week 4: Is There a Difference between Client Side and Server Side?" (https://n3tweb.wordpress.com/2015/10/15/week-4-is-there-a-difference-between-client-side-and-server-side/). *n3tweb.wordpress.com*. Retrieved 2017-07-07.
10. Espinosa, Christian (2016-04-23). "Decoding the Hack" (https://www.alpinesecurity.com/s/Alpine-Security-Decoding-the-Hack-Presentation-22-April-16.pdf) (PDF). *alpinesecurity.com*. Retrieved 2017-07-07.
11. Rulifson, Jeff (June 1969). *DEL* (https://datatracker.ietf.org/doc/html/rfc5). IETF. doi:10.17487/RFC0005 (https://doi.org/10.17487%2FRFC0005). RFC 5 (https://datatracker.ietf.org/doc/html/rfc5). Retrieved 30 November 2013.
12. Shapiro, Elmer B. (March 1969). *Network Timetable* (https://datatracker.ietf.org/doc/html/rfc4). IETF. doi:10.17487/RFC0004 (https://doi.org/10.17487%2FRFC0004). RFC 4 (https://datatracker.ietf.org/doc/html/rfc4). Retrieved 30 November 2013.
13. Sturgis, Howard E.; Mitchell, James George; Israel, Jay E. (1978). "Separating Data from Function in a Distributed File System" (http://ip.com/IPCOM/000128883). Xerox PARC.
14. Harper, Douglas. "server" (https://www.etymonline.com/?term=server). *Online Etymology Dictionary*. Retrieved 30 November 2013.
15. "Separating data from function in a distributed file system" (https://web.archive.org/web/20131202233729/https://getinfo.de/app/Separating-data-from-function-in-a-distributed/id/TIBKAT%3A509976956). *GetInfo*. German National Library of Science and Technology. Archived from the original (https://getinfo.de/app/Separating-data-from-function-in-a-distributed/id/TIBKAT%3A509976956) on 2 December 2013. Retrieved 29 November 2013.
16. Nieh, Jason; Yang, S. Jae; Novik, Naomi (2000). "A Comparison of Thin-Client Computing Architectures" (https://academiccommons.columbia.edu/doi/10.7916/D8Z329VF). *Academic Commons*. doi:10.7916/D8Z329VF (https://doi.org/10.7916%2FD8Z329VF). Retrieved 28 November 2018.
17. d'Amore, M. J.; Oberst, D. J. (1983). "Microcomputers and mainframes". *Proceedings of the 11th annual ACM SIGUCCS conference on User services - SIGUCCS '83*. p. 7. doi:10.1145/800041.801417 (https://doi.org/10.1145%2F800041.801417). ISBN 978-0897911160. S2CID 14248076 (https://api.semanticscholar.org/CorpusID:14248076).
18. Tolia, Niraj; Andersen, David G.; Satyanarayanan, M. (March 2006). "Quantifying Interactive User Experience on Thin Clients" (https://www.cs.cmu.edu/~dga/papers/tolia06-ieee.pdf) (PDF). *Computer*. **39** (3). IEEE Computer Society: 46–52. doi:10.1109/mc.2006.101 (https://doi.org/10.1109%2Fmc.2006.101). S2CID 8399655 (https://api.semanticscholar.org/CorpusID:8399655).
19. Otey, Michael (22 March 2011). "Is the Cloud Really Just the Return of Mainframe Computing?" (https://web.archive.org/web/20131203011958/http://sqlmag.com/cloud/cloud-really-just-return-mainframe-computing). *SQL Server Pro*. Penton Media. Archived from the original (http://sqlmag.com/cloud/cloud-really-just-return-mainframe-computing) on 3 December 2013. Retrieved 1 December 2013.
20. Barros, A. P.; Dumas, M. (2006). "The Rise of Web Service Ecosystems". *IT Professional*. **8** (5): 31. doi:10.1109/MITP.2006.123 (https://doi.org/10.1109%2FMITP.2006.123). S2CID 206469224 (https://api.semanticscholar.org/CorpusID:206469224).
21. Cardellini, V.; Colajanni, M.; Yu, P.S. (1999). "Dynamic load balancing on Web-server systems". *IEEE Internet Computing*. **3** (3). Institute of Electrical and Electronics Engineers (IEEE): 28–39. doi:10.1109/4236.769420 (https://doi.org/10.1109%2F4236.769420). ISSN 1089-7801 (https://www.worldcat.org/issn/1089-7801).

22. "What Is Load Balancing? How Load Balancers Work" (https://www.nginx.com/resources/glossary/load-balancing/). *NGINX*. June 1, 2014. Retrieved January 21, 2020.
23. Varma, Vasudeva (2009). "1: Software Architecture Primer" (https://books.google.com/books?id=jOMYtrJ6r_0C). *Software Architecture: A Case Based Approach*. Delhi: Pearson Education India. p. 29. ISBN 9788131707494. Retrieved 2017-07-04. "Distributed Peer-to-Peer Systems [...] This is a generic style of which popular styles are the client-server and master-slave styles."

▪