Module_6 - Client/Server Architecture

6.1 - Introduction and Background to Client/Server Systems and Multi-tier Architecture

The client/server model is the standard model of networked traffic in today's computing and database systems. The client/server model was designed with world wide web services in mind and to allow for servers to provide service to end user devices all over the network. This methodology has been adopted as the standard for content publishing and receiving by modern network systems.

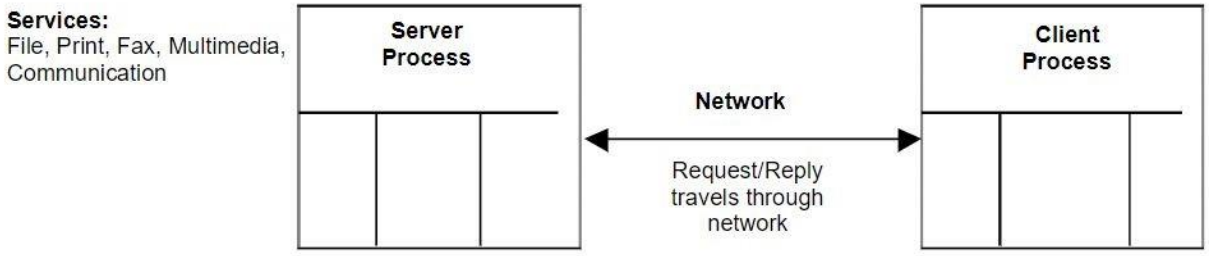

Figure 1. A graphic demonstrating the architecture of client/server systems. From "*An Introduction to Client/Server Computing*" by Yadav SC, Singh SK, 2009.

For databases, the model of client/server is ideal because the database can easily serve multiple users at the same time through standardized calls for information. If multiple users are looking to fetch the same files on a routine basis, tools can be deployed which monitor the trends of usage and allow for IT administrators to beef up the systems which serve those files, reducing wait times and increasing overall system performance. Similarly, for systems or databases which are accessed less frequently, those costs can be avoided.

Prior to the client/server architecture, peer-to-peer networking was used. This means that for every request to send or receive, devices would communicate one-to-one, and for each transaction of data, this process had to be repeated. By taking on the role of a "server", a system can handle more than one call at once, drastically improving throughput and efficiency. Using the client/server model gives rise to a very specific set of requirements when designing a transport layer protocol. Generally, a transport layer protocol needs to cover all of the following requirements: a connection between a client and server, an interaction between the client and server, authentication of the client and server, and a checksum or other method to maintain data integrity after having sent and received packets.

Multi-Tier Architecture (MTA) can be generalized as a separation and duplication of a server/database system by decentralizing data and compute resources. MTA is used to improve reliability and throughput, similarly to client/server architecture. As such, an MTA environment will almost certainly use client/server communication.

MTA systems are composed of 6 basic layers (Wall, D, Morgan Kaufmann):

- Persistence: The database which serves files to the applications that service the clients or users.
- Accessor: Typically, the SQL server. Not the disks in the database, but the part of the database which does the thinking.

- Logic: The applications which the users interact with, in turn providing instructions to the database from which files are requested.
- Presentation: Systems applications which package data from the database into web browser languages such as HTML or XML.
- Requester/consumer: The web browser itself; this is on the client side.
- Elsewhere: The sources of information hosted on other platforms, such as AWS, Azure, or GCP, or other sources of HTML content that are not native.

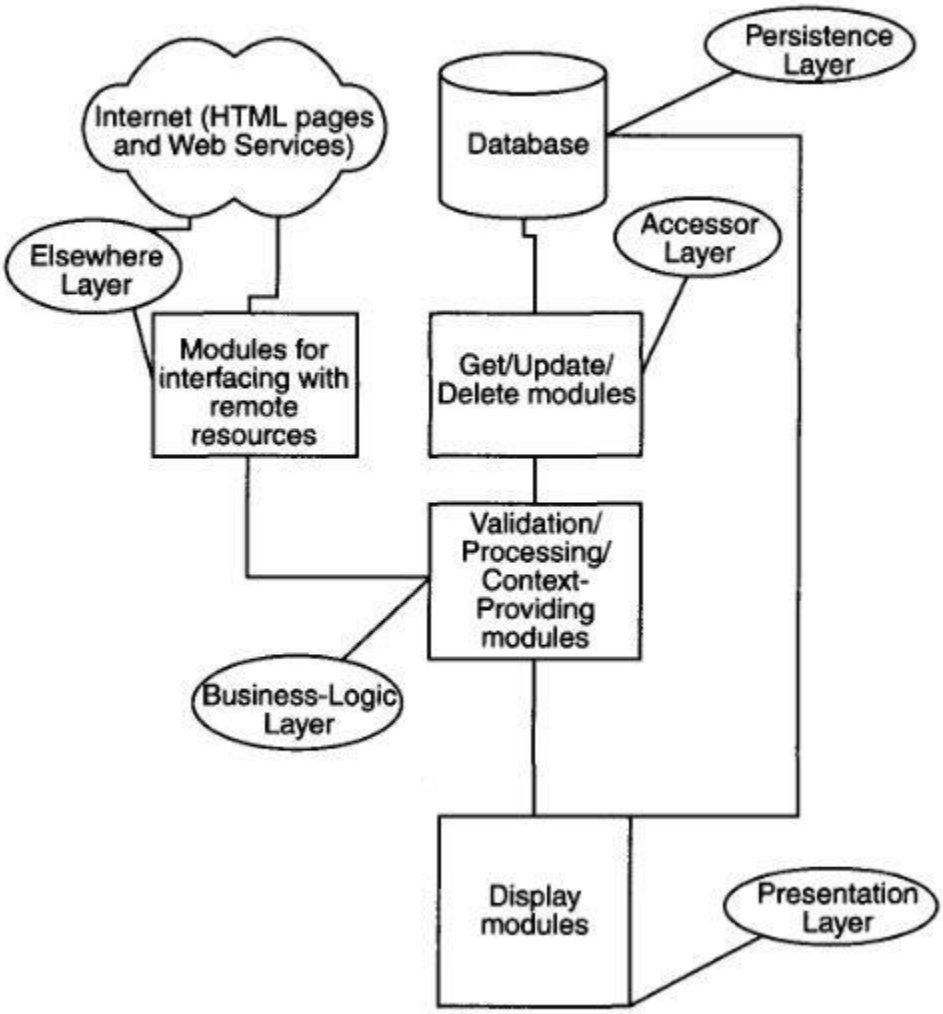

Figure 2. A graphic demonstrating the architecture of multi tier architecture. From "*Multi-Tier Application Programming with PHP: Practical Guide for Architects and Programmers"* by Wall, D, Morgan Kaufmann, 2004.

6.2 - Three components of client/server systems

The client/server model is designed to manage data and facilitate the methods by which users can access it. In the client/server model, data is shared between one central server that houses and provides access to it and one or more clients that access the data through either a direct or remote feed. By calling requests to the server, clients can access any information they have access to, but clients cannot access information from other clients, as that would represent a peer-to-peer system, rather than a client/server model.

In client/server systems, there are three main components of logic. Presentation logic controls the inputs and outputs. It "presents" the data to the user in a readable format and also collects information that a user inputs using their system. Processing logic oversees everything around turning the client request into a request the server can use in the next step to find information. The final component is storage logic, which handles physical data storage and processes for retrieving it upon user request.

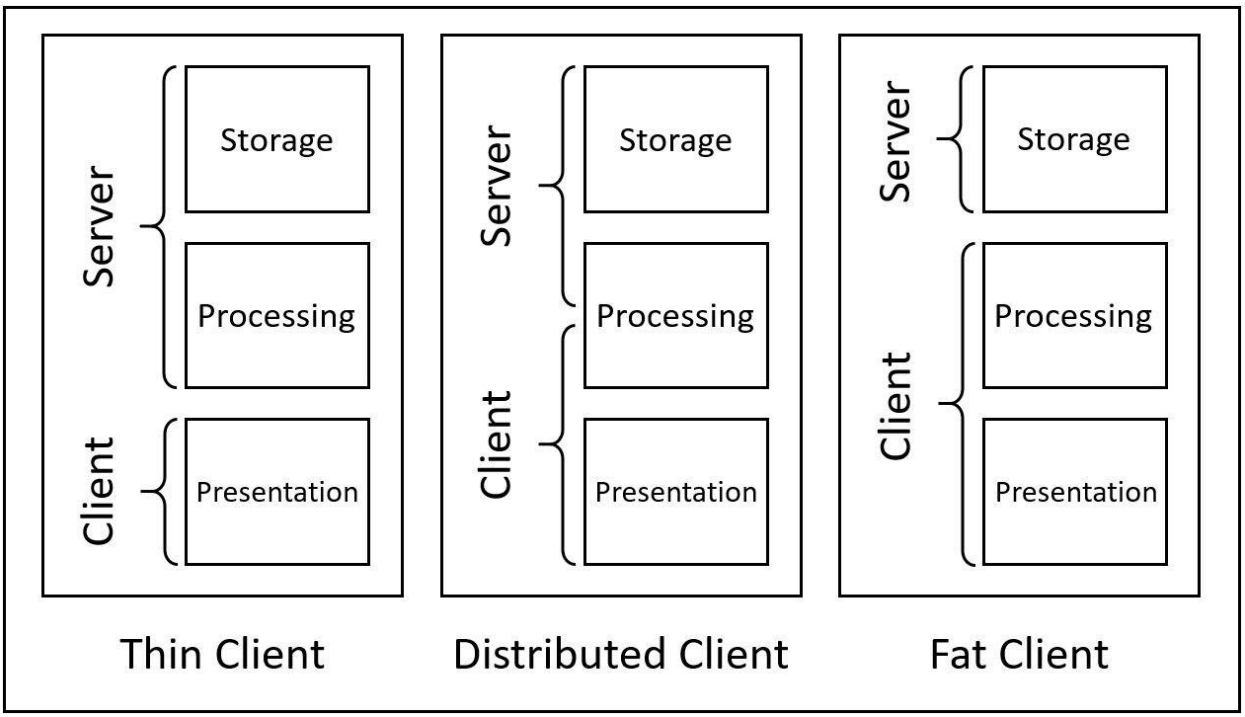

Figure 3. Client distribution models (made via Microsoft PowerPoint, Michael Nolan 2020)

The way that systems using this model delegate these logical applications defines individual databases further. Two-tier database architecture is divided into three groups named to specify what the client takes care of, as shown in Figure 1. In fat client distribution, presentation and processing are both handled by the client system, while the server exclusively handles the storage logic application. In thin client distribution, the client only handles presentation, while the server takes care of processing the request and dealing with storage. Finally, when a two-tier system is distributed in design, the client and server both handle parts of processing logic. In these sorts of systems, presentation logic is still handled by the client and storage is still handled by the server exclusively.

Further levels of tiered architecture in databases works similarly to the aforementioned design ideas, but splits processing logic applications further between multiple intermediate applications and web servers before the client's request reaches the database server.

a. Presentation

Presentation logic is the high-level interactive portion of the database. It is where the end-user can input requests and receive output on their host device.

Input in the presentation level of the client-server model can be many different things. GUI elements can be added to make it easier, such as text fields that users can modify, copy, or delete, buttons for submitting requests, menus, links, and more. Output is handled by the system to present the server response to user queries in a readable way that gives the user all the information they need without giving them information they did not request or information to which they are not allowed access.

Presentation logic is handled exclusively by client software on the host device. This can be managed by some language such as HTML to quickly take inputs and display outputs in a controlled location.

b. Processing

The processing logic layer is an intermediary layer that translates information back and forth between user language coming from the client software and abstracted language used by the database's storage unit to quickly access the correct information. The processing layer also handles business rules, assuming they are not already built into the organizational DBMS.

Processing logic has many automated measures defined by business logic and business rules that are tailored to the specific company or organization the database is a part of. It is also where data integrity is insured whenever commands are sent to the storage database.

Applications for this are generally coded in Java or C, as those languages can be applied to many kinds of hardware with ease and offer some scalability without slowing down the system. Attention needs to be paid to these processes to make sure that this stage of data transfer does not bottleneck, as the processing logic may have to handle many requests per day.

Processing logic is also where the majority of differences in database architecture lie. Two-tier architecture can be split three ways: fat client, thin client, and distributed. Fat client distribution has all processing logic handled on the client's side, so the user host machine sends a request to the database having already translated the language from the user's input into whatever language that specific database uses for queries. Fat client distribution is the most commonly used organization of database access in the client-server system. Thin client distribution has all processing logic handled by the server system. The goal of this is to keep the user interface as light as possible. Distributed systems have processing logic handled by both the client and server in a way that boosts efficiency. It is more complex than the other two methods to set up.

Multi-tier organization splits processing logic in different ways. Three-tier systems may have processing logic all performed on a separate web server that clients and servers both access. Above that, processing can be handled by multiple web and application servers that each manage smaller parts of the translation. The goal of these larger systems is to be more modular and scalable while still being easy to maintain, since problems can be pinpointed easily.

c. Storage

The storage logic represents the physical database and storage devices. It is where data storage and retrieval is handled in the system, and it is the part of the client-server system where the DBMS interacts with everything else.

Data in the database can be stored in many different ways. Data can be accessed using the information provided by the processing applications very quickly.

6.3 Two-tier and three-tier architectural distinctions

Though many people may not realize it, when a group of tech-savvy individuals develops software, one of the most important decisions at the start is the architecture: “architecture is a key feature which decides that the system will meet its performance and other quality objectives” (Hayat and Akram, 2007). Several main components come into play when it comes to deciding the type of architecture, including performance, availability, complexity, and cost.

Performance depends on the quantity of simultaneous users that can be processed by the database system to ensure smooth processing. Typically, most applications limit the number of users accessing different services of the application by only providing them with restricted data. While choosing an architecture, it is quite important to keep in mind the performance component. Another significant factor is the cost. The selected architecture must be affordable for any organization just starting. Availability is the accessibility of the system to the community of users. Complexity is the extent of difficulty of the system in terms of the user’s and developer’s viewpoint. If the developers feel they want a lower level of difficulty, they can make the user’s level of difficulty rise due to lack of user interface organization/improvements. However, to lower the user’s level of difficulty, the developers may have to increase their effort and continuously improve, revise, and contemplate new ways to make the user interface easier and more accessible.

In two-tier architectures, the interfaces run on a client, which means the client communicates with the server directly, and because it runs on a client, another data layer or structure is then stored on a server, as seen in Figure 4 below. The client's responsibility covers user interface logic from the presentation, which deals with how objects in the business get shown to other users of the software as well as data processing and rules (algorithms) that dictate the information that gets exchanged between the database system and the user interface. Meanwhile, database servers access and process requests from the clients. This architecture allows for optimization of the processing time in the database servers and is highly recommended for small workgroups. However, clients or servers that use two-tier architecture tend only to support a relatively small number of users, because the transaction of information is usually low, and security is not necessarily the top priority. Because of this, two-tier architectures are not used for applications with large user bases due to their limited flexibility.

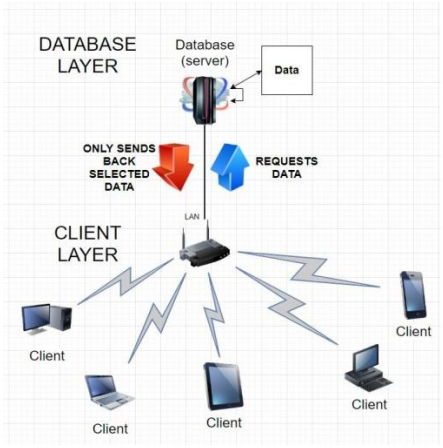

Figure 4 Visual of two-tier architecture (two layers)
(Made via Draw.io, Martin Nguyen, 2020)

In general, three-tier (n-tier) architecture includes another server layer, as seen in Figure 2. This extra layer make it much easier to scale the system, which makes performance much better, resulting in a highly flexible and reusable structure. The extra layer can have several purposes, such as an application server, or it can be used as local storage and hold local databases. In contrast, the other servers hold more enclosed data. Utilizing a three-tier architecture can lead to better performance, and make it easier to transport application code to other platforms, reducing dependency on commonly domain-specific language. Three-tier systems can often be found in Web-based applications—any program accessible using HTTP.

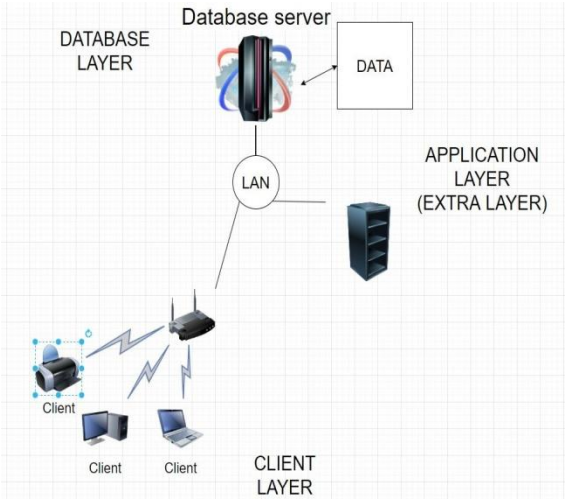

Figure 2 Visual of three-tier (three or nth layers) (Made via Draw.io, Martin Nguyen, 2020)

6.4 Connecting to databases in three-tier applications

Three-tier application structure is similar to the 7-layer OSI model of telecommunications, in that it breaks up the presentation, data, and application functions into each of their own levels. This segmentation provides for a more streamlined approach to scalability and ease of development, since the developer does not have to produce a new web technology for every instance. Instead, standards such as HTML, CSS, or JavaScript may be used.

When connecting to a three-tier based web application, the user will interact with the presentation tier. This layer is the visual wrapper between the functionality, database, and UI elements. This layer is displayed in a web browser, and served through the World Wide Web or a Local Area Network. Connecting to an IP address using a compatible browser will provide the web page to the user, being provided through a server which hosts the website and has access to the databases which it uses. The presentation tier is largely the visual aspect of a website or application and does not perform the computing or data storage/retrieval. This layer is important, however, because data entry and user intent must be clearly interpreted for the right API calls to be made by the presentation layer, to be sent to the application and data tiers.

By using a three-tier architecture, developers are able to concurrently work on each tier of the application as they are designing it. This parallel development allows for fewer limitations due to older design which cannot easily be changed. This is relevant to the connection to databases, due to the ability of the development teams to agree on which APIs and methods they wish to communicate with between each layer. For the database to be accessed, the user would

make a request for a piece of information to be pulled from the database through the presentation layer. The presentation layer would process the request through the application layer (or logic layer) which would request the data from the database. All of this would happen through the use of API calls in a modern application. The database layer would be a version of SQL such as MySQL, Microsoft SQL Server, or another database type.

Once the request reaches the data layer, the database management system (DBMS) is what will actually do the work of finding what the request is looking for. This includes querying the database indexes to see where the information is stored, as databases are usually composed of one or more storage devices with many logical storage endpoints.

6.5 - Section Concise Summary

The client-server model is regularly used today for designing system architecture. At its base, it is a simple back-and-forth transfer of information from client requests to server responses, but it is highly customizable for many functions.

Different client-server implementations can take several forms, depending on how many tiers exist in the design. They range from simple thin and fat client two-tier systems to modular distributed systems, up to any number of tiers of web and application servers separating clients from their servers. The addition of layers of abstraction serves to make the host client's UI as simple to use as possible, making them desirable to many end users.

6.6 - Description & Links

1. Gives the definition and a brief background of client server model/architecture. Explains the topics of advantages and disadvantages of the model as well as protocols and other program relationship models as well. The video goes through and describes as well as gives a visual of client server architectures.
   https://searchnetworking.techtarget.com/definition/client-server

2. This video talks about software multilayered architectures consisting of one tier, two tier, three tier, and n-tier architectures.
   https://www.youtube.com/watch?v=ccOzcjHQVo_s

3. This slideshow walks you through and visually shows you how to connect database files to tier architecture applications.
   https://www.slideshare.net/fkbullet11/con-stringslideshare

4. A history of the client server.
   https://pdfs.semanticscholar.org/f47d/f7345d0e5bdf5500c5f27a4ba92ac400685a.pdf

5. Microsoft's client/server application guidelines.
   https://docs.microsoft.com/en-us/windows/win32/termserv/client-server-application-guidelines

6. A detailed overview of client server applications.
   https://www.sciencedirect.com/topics/computer-science/client-server-application

7. This video gives a basic tutorial on client server applications.
   https://www.youtube.com/watch?v=3FKHKiNVYI_4

6.8 - Reference List

Akram, Muhammad. “Qualitative & Quantitative analysis of tiered Architecture of Web-Applications.“ 2007. PDF.

Yadav SC, Singh SK. An Introduction to Client/Server Computing. New Delhi: New Age International; 2009.

Wall, D, Morgan Kaufmann. Multi-Tier Application Programming with PHP : Practical Guide for Architects and Programmers; 2004.