Chapter 1 : Client-Server Architecture

The client-server model 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.

This three-part series is a general, high-level, first-day-of-Intro-to-Web-Development overview of web app architecture. There are a multitude of tutorials on how to do tiny, specific, web-dev-related tasks; but surprisingly few tutorials, explaining how all of the little pieces are supposed to fit together. The browser goes out and finds the internet-facing computer that the website lives on and asks the server for the specific page. The server responds to the request by sending some files over to the browser. The browser executes those files and shows something hopefully! The user can now visually interact with the website. The code that has been parsed by the browser may or may not have instructions telling the browser to react to user input in various ways: Client Side The main thing to take away from the last paragraph is that in a web application, there are basically two programs running at the same time: The code that lives on the server and responds to HTTP requests. The code that lives in the browser and responds to user input. That isâ€¦ not the case here. Deciding what the code on the server should do versus what the code on the browser should do are just a few of the hundreds of things a web developer has to decide when they start to write their app. But in general, server and browser decisions tend to be split up like this: Any code that can run on a computer and respond to HTTP requests can run a server. Stores persistent data user profiles, instatweets, mybook pages, etc. Cannot be seen by the user unless something is terribly wrong. Creates the page that the user finally sees this is generally only true in web applications that choose to render most of their layouts on the server. Client-Side Code Languages used include: Reacts to user input. Can be seen and edited by the user in full. Cannot store anything that lasts beyond a page refresh. Cannot read files off of a server directly, must communicate via HTTP requests. Creates the page that the user finally sees this is generally only true in single page applications. Bringing them Together The generally accepted practice for building a web applicationâ€"as in an application accessible via a browser that mimics a desktop app, not a static websiteâ€"is to have your server deliver your homepage and handle saving and loading persistent data, based on simple messages, from the browser aka making a RESTful API, more on this later. This general style of app is called a Single Page Application. This is the first in a series on understanding SPA architecture:

Chapter 2 : Differences Between Thick and Thin Clients -Webopedia

The client application will have Multiple Document Interface. Below are my points that what i am thinking about - The client application will use socket connectivity with server application so as to get authenticated from server, and also to track record of client connections over server.

Sign in to vote Concerning the validation, if it is global in nature, meaning everyone will use it, then that is a good place for it at that tier. But if others will have different validations, it might be better to not validate it at that level, but have the the validation done at the GUI or other tiers. Let me explain, since the other tiers have unique needs then they should derive a new class from Person and have its validation requirements done there locally in that tier. Or a combination of both, where one level specifies a validation requirement but calls the base level which may have it own. To use our system of age, one can vote at 18, but buy beer at The base could check for 18, but the GUI could check for Or if the person lives in Canada I would try to avoid using exceptions unless its a dire and unplanned for situation, such as out of memory or database unaccessible. With the situation where a unique name rule needs to be enforced, I would recommend that the objects provide a status with the data they are returning. Its the responsibility of each tier to check the status and appropriate return a status to the next tier. For example, my business layer needed to do similar checks. I had different status, one such status was a general status such as operation succeeded or failed. Then I would have a secondary status, usually as an enum which gave a failure indicator. The user could get an error message, derived from the enum see my blog entitled C Enum Extended Attribute Information as to how I did that which was displayed to the user. That way you handle all errors in a consistent manner for all levels. This is a bit of work, but once in place, it flows. Create an enum which specifies errors. Create a methodology where data is returned with a state an enum of status for each tier. Make sure each tier handles errors appropriately and passes the info up within that methodology. One other thing, your example has it, but have the tiers talk to each other using Interfaces. That way if you ever need to change the back end processing, the subsequent tier will not be affected. I put all my enums and interfaces into one non object assembly. That way each tier has access to the interfaces but not objects which could complicate the tier system. Monday, July 2, 3:

Chapter 3 : Client/Server Application Architecture

Client-server architecture (client/server) is a 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).

Image via Wikimedia Commons The separate physical location of these tiers is what differentiates n-tier architecture from the model-view-controller framework that only separates presentation, logic, and data tiers in concept. N-tier architecture also differs from MVC framework in that the former has a middle layer or a logic tier, which facilitates all communications between the different tiers. When you use the MVC framework, the interaction that happens is triangular; instead of going through the logic tier, it is the control layer that accesses the model and view layers, while the model layer accesses the view layer. Additionally, the control layer makes a model using the requirements and then pushes that model into the view layer. This is not to say that you can only use either the MVC framework or the n-tier architecture. There are a lot of software that brings together these two frameworks. For instance, you can use the n-tier architecture as the overall architecture, or use the MVC framework in the presentation tier. What are the Benefits of N-Tier Architecture? There are several benefits to using n-tier architecture for your software. These are scalability, ease of management, flexibility, and security. You can secure each of the three tiers separately using different methods. You can manage each tier separately, adding or modifying each tier without affecting the other tiers. If you need to add more resources, you can do it per tier, without affecting the other tiers. Apart from isolated scalability, you can also expand each tier in any manner that your requirements dictate. In short, with n-tier architecture, you can adopt new technologies and add more components without having to rewrite the entire application or redesigning your whole software, thus making it easier to scale or maintain. Meanwhile, in terms of security, you can store sensitive or confidential information in the logic tier, keeping it away from the presentation tier, thus making it more secure. N-tier architecture is very friendly for development, as different teams may work on each tier. This way, you can be sure the design and presentation professionals work on the presentation tier and the database experts work on the data tier. Easy to add new features. If you want to introduce a new feature, you can add it to the appropriate tier without affecting the other tiers. Because the application is divided into independent tiers, you can easily reuse each tier for other software projects. For instance, if you want to use the same program, but for a different data set, you can just replicate the logic and presentation tiers and then create a new data tier. In this setup, you have the presentation or GUI tier, the data layer, and the application logic tier. The application logic tier. This logic tier is also the one that writes and reads data into the data tier. The data tier is where all the data used in your application are stored. You can securely store data on this tier, do transaction, and even search through volumes and volumes of data in a matter of seconds. The presentation tier is the user interface. This is what the software user sees and interacts with. This is where they enter the needed information. Just imagine surfing on your favorite website. The presentation tier is the Web application that you see. If you need to log in, the presentation tier will show you boxes for username, password, and the submit button. After filling out and then submitting the form, all that will be passed on to the logic tier. The logic tier would be run on a Web server. All of these are run on a separate database server. Rich Internet applications and mobile apps also follow the same three-tier architecture. And there are n-tier architecture models that have more than three tiers. Examples are applications that have these tiers: Presentation tier Client tier â€” or the thin clients One good instance is when you have an enterprise service-oriented architecture. The enterprise service bus or ESB would be there as a separate tier to facilitate the communication of the basic service tier and the business domain tier. Considerations for Using N-Tier Architecture for Your Applications Because you are going to work with several tiers, you need to make sure that network bandwidth and hardware are fast. Also, this would mean that you would have to pay more for the network, the hardware, and the maintenance needed to ensure that you have better network bandwidth. Also, use as fewer tiers as possible. Remember that each tier you add to your software or project means an added layer of complexity, more hardware to purchase, as well as higher maintenance and deployment costs. To make your n-tier applications make sense, it should have the minimum number of tiers needed to still enjoy the scalability, security and other benefits brought about by using this architecture. N-Tier Architecture Tutorials and Resources For more information on n-tier architecture, check out the following resources and tutorials:

Chapter 4 : Difference Between Client Server Application and Web Application | Difference Between

Hi, I have a question. I'm trying to understand application architecture, you know, n-tiers/n-layers. If I have to design an application client-server where the server will have an UI tier, Business Objects tier that will move data from UI to Database and from Database to UI, and Data tier, to wich tier or layer will be part the "network" stuff, I mean, the server will listent for client.

The trend of business process reengineering has generated the need to improve communications, reduce overhead, enhance work-process efficiencies and facilitate information sharing across departmental and organizational boundaries. New Business Requirements Recent government initiatives to expedite the purchase ordering process, improve inventory control and deliver better services to the public have created demands for applications that would link up the government agencies to their vendors, partners and customers. In addition, not only is multi-platform support essential, these applications also have to be adaptable to emerging client operating systems e. Finally, considerations have to be made for dial-up or remote users. A stored procedure is written in a vendor-specific SQL dialect. It resides in a database for processing data requests sent from clients. Applications created using this approach generally have a two-tier architecture. The first tier is a single process running on the user machine. The second tier is the database server with stored procedures residing in it. Each user connection takes up some resources on the database server. As the number of users increases, the single database server will sooner or later run out of resources. More database servers will be required to accommodate the growing install base. One way to solve the problem is with a two-tier model and database replication. By replicating data on one server to another server, the number of users supported can be doubled. Data replication, however, is only suited for certain types of applications - those with a low chance of simultaneous record updates. In other words, when it is unlikely that two or more users will be updating the same record at the same time, data replication could be an appropriate approach to solving the scalability problem. However, for those applications with a high volume of transactions and with potentially hundreds or thousands of users, one may wish to find an alternative solution. Unlike a two-tier model, it may or may not contain the application logic. The second tier is an object server that holds part or all of the application logic. Finally, the database forms the third tier. As mentioned above, every single user in a two-tier model has to maintain a connection with the database. Resources on the database server will soon be exhausted as the number of users continues to grow. In a three-tier model, on the other hand, only one connection has to be maintained between the object server and the database. All data requests from the clients will be routed through the object server which acts as a request broker. In this scenario, the system can be scaled to accommodate more users simply by adding more object servers. The "thin" client approach is a special case of the three-tier architecture. A thin client contains modules to create user interfaces U. For example, a Macintosh client has Mac-specific U. User interfaces generated by these modules can perform field type-checking and compute calculated fields that have values dependent on the values of other fields, e. A thin client also means a "generic" client as it only contains code that can be used by any application. All application logic resides on the object server which, in response to user requests, gives instructions to the client by sending messages on how the screen should look, how the fields should be defined and what values should populate the fields. As a result, unlike a "fat" client which bundles application and presentation logic into a single executable , a thin client can run different applications without a single change on the user workstation. All that have to be modified are the instructions sent down to the client from the server. The thin-client architecture can better manage remote users and help save significant application administration costs. In an environment where there are lots of remote users, distributing new applications or version upgrades can be quite time-consuming and costly. Under the two-tier model, the new code has to be downloaded or installed onto every single client machine. With the thin-client approach, fortunately, no change is necessary on the user workstations. Moreover, the thin-client design helps to insulate your application development investment from future operating system changes on the client side. This benefit is attributed to two factors. First, the application modules are processed on the object server. Second, the instructions sent to the client are platform-independent As a result, a user who wishes to switch from a Macintosh to a Windows machine, for example, can do so without any change to the application which resides on the object server. Under a three-tier, thin-client architecture - in order to ensure reasonable performance, system modularity and ease of maintenance - an application should be made up of many small transactions. An object server has the capability to send requests to databases as well as other object servers. As a consequence, with careful initial design, future growth in transaction volume can be handled incrementally by adding additional object servers. In order to build a system with a three-tier, thin-client architecture, information technology. These tools are already available in the industry today.

Chapter 5 : Client-server architecture | computer science | theinnatdunvilla.com

1. Client server application using two tier architecture. using client and server only. changes can't reflect in this. used to install again and test whether application works or not.

A server component perpetually listens for requests from client components. When a request is received, the server processes the request, and then sends a response back to the client. Servers may be further classified as stateless or stateful. Clients of a stateful server may make composite requests that consist of multiple atomic requests. This enables a more conversational or transactional interactions between client and server. To accomplish this, a stateful server keeps a record of the requests from each current client. This record is called a session. In order to simultaneously process requests from multiple clients, a server often uses the Master-Slave Pattern. In this case the Master perpetually listens for client requests. When a request is received, the master creates a slave to processes the request, and then resumes listening. Meanwhile, the slave performs all subsequent communication with the client. Here is a simple component diagram showing a server component that implements operations specified in a Services interface, and a client component that depends on these services. Internally, the client component may consist of a ClientUI that forwards user requests to a controller component. The controller component forwards the request across a process or machine boundary to a RequestListener inside the server. The listener, which acts like a master, creates a RequestHandler slave and forwards the request to it: The following sequence diagram shows a typical client-server interaction: The obvious way to do this would be to create a derived class with the added features: Instead, we must create new servers that instantiate the derived class. Another approach uses the idea of the Decorator pattern introduced in Chapter 4. Instead of creating a server object that instantiates a derived class, we place a decorator-like object called a proxy between the client and the server. The proxy intercepts client requests, performs the additional services, then forwards these requests to the server. The server sends any results back to clients through the proxy. Of course the proxy has no way of knowing if the server it delegates to is the actual server or just another proxy in a long chain, and of course the server has no way of knowing if the object it provides service to is an actual client or a proxy. The idea is formalized by the Proxy design pattern: They are closely related in structure, but not purpose, to adapters and decorators. A server may provide the basic services needed by clients, but not administrative services such as security, synchronization, collecting usage statistics, and caching recent results. Inter-process communication mechanisms can introduce platform dependencies into a program. They can also be difficult to program and they are not particularly object-oriented. Structure To simplify creation of proxies, a Proxy base class can be introduced that facilitates the creation of delegation chains: The class diagram suggests that process or machine boundaries may exist between proxies. Proxies that run in the same process or on the same machine as the client are called client-side proxies, while proxies that run in the same process or on the same machine as the server are called server-side proxies. Scenario As in the decorator pattern, proxies can be chained together. The client, and each proxy, believes it is delegating messages to the real server: Examples of Client-Side Proxies A firewall proxy is essentially a filter that runs on the bridge that connects a company network to the Internet. It filters out client requests and server results that may be inconsistent with company policies. A cache proxy is a client-side proxy that searches a local cache containing recently received results. If the search is successful, the result is returned to the client without the need of establishing a connection to a remote server. Otherwise, the client request is delegated to the server or another proxy. For example, most web browsers transparently submit requests for web pages to a cache proxy, which attempts to fetch the page from a local cache of recently downloaded web pages. Virtual proxies provide a partial result to a client while waiting for the real result to arrive from the server. For example, a web browser might display the text of a web page before the images have arrived, or a word processor might display empty rectangles where embedded objects occur. Examples of Server-Side Proxies Protection proxies can be used to control access to servers. It demands user identifications and passwords before it forwards client requests to the web server. A synchronization proxy uses techniques similar to the locks discussed earlier to control the number of clients that simultaneously access a server. High-volume servers run on multiple machines called server farms. A load balancing proxy is a server-side proxy that keeps track of the load on each server in a farm and delegates client requests to the least busy server. Counting proxies are server-side proxies that maintain usage statistics such as hit counters. Remote Proxies and Remote Method Invocation Communicating with remote objects through sockets is awkward. It is entirely different from communicating with local objects, where we can simply invoke a member function and wait for a return value. The socket adds an unwanted layer of indirection, it restricts us to sending and receiving strings, it exposes the underlying communication protocol, and it requires us to know the IP address or DNS name of the remote object. A remote proxy encapsulates the details of communicating with a remote object. This creates the illusion that no process boundary separates clients and servers. Clients communicate with servers by invoking the methods of a client-side remote proxy. Servers communicate with clients by returning values to server-side remote proxies. For example, a client simply invokes the member functions of a local implementation of the server interface:

Chapter 6 : Client Server Architecture Advantages & Disadvantages | theinnatdunvilla.com

Web Application Architecture from 10, Feet, Part 1 - Client-Side vs. Server-Side Or, Why you can't get your jQueryUI Datatables plugin to keep your data after you refresh the page. This three-part series is a general, high-level, first-day-of-Intro-to-Web-Development overview of web app architecture.

Share on Facebook As technology and computers have continued to develop rapidly, a client server network has soon replaced past forms of networking on a computer to become the most widely used. A client server network can be utilized by desktop computers and laptops, as well as other mobile devices that are properly equipped. Identification A client server network is defined as specific type of online network comprised of a single central computer acting as a server that directs multiple other computers, which are referred to as the clients. By accessing the server, clients are then able to reach shared files and information saved on the serving computer. Further, client server networks are very similar in nature to peer to peer networks with the exception that it is only the server that can initiate a particular transaction. Features A client server model can be implemented into a single computer system, but is most commonly applied over many different sites. This makes it possible for multiple computers or people to interconnect and share information. Video of the Day As businesses expand and people are now working together across vast distances, a client server model enables them to reach a common, or shared, database or program. This works as well when online users access their bank account or pay particular bills online. Benefits The main benefits of the client server network is allowing a shared database or site to be accessed or updated by multiple computers while maintaining only one control center for the action. This makes it possible for companies to distribute information, upload data, or reach the program without being tied down to one individual computer site. Because the information is stored online, a client server model creates more power and control over what is being saved. Additionally, this model has an increased security, often with encryption, ensuring that the data is only available to qualified individuals. Warnings Under a client server model, the main disadvantage is running the risk of a system overload. If too many different clients attempt to reach the shared network at the same time, there may be a failure or a slowing down of the connection. Furthermore, if the network is down, this disables access to the information from any site or client anywhere. This can be detrimental to major businesses who are unable to reach their pertinent data. Considerations Other types of service connections include master slave networks and peer to peer networks. In a master slave diagram, a single program is in charge of all the others, with one being dominant over the other. In contrast, a peer-to-peer network, while similar to a client-server architecture, differentiates in that it allows any client to start transactions.

Chapter 7 : Client/Server Architecture

Client/server architecture is a producer/consumer computing architecture where the server acts as the producer and the client as a consumer. The server houses and provides high-end, computing-intensive services to the client on demand.

Common layers[edit] In a logical multilayered architecture for an information system with an object-oriented design , the following four are the most common: UI layer, view layer, presentation tier in multitier architecture Business layer a. This layer is very general and can be used in several application tiers e. In other words, the other kind of technical services are not always explicitly thought of as part of any particular layer. Every layer can exist without the layers above it, and requires the layers below it to function. Another common view is that layers do not always strictly depend on only the adjacent layer below. For example, in a relaxed layered system as opposed to a strict layered system a layer can also depend on all the layers below it. Three-tier architecture is a clientâ€"server software architecture pattern in which the user interface presentation , functional process logic "business rules" , computer data storage and data access are developed and maintained as independent modules , most often on separate platforms. Apart from the usual advantages of modular software with well-defined interfaces, the three-tier architecture is intended to allow any of the three tiers to be upgraded or replaced independently in response to changes in requirements or technology. For example, a change of operating system in the presentation tier would only affect the user interface code. Typically, the user interface runs on a desktop PC or workstation and uses a standard graphical user interface , functional process logic that may consist of one or more separate modules running on a workstation or application server , and an RDBMS on a database server or mainframe that contains the computer data storage logic. The middle tier may be multitiered itself in which case the overall architecture is called an "n-tier architecture". Presentation tier This is the topmost level of the application. The presentation tier displays information related to such services as browsing merchandise, purchasing and shopping cart contents. Data tier The data tier includes the data persistence mechanisms database servers, file shares, etc. The data access layer should provide an API to the application tier that exposes methods of managing the stored data without exposing or creating dependencies on the data storage mechanisms. Avoiding dependencies on the storage mechanisms allows for updates or changes without the application tier clients being affected by or even aware of the change. As with the separation of any tier, there are costs for implementation and often costs to performance in exchange for improved scalability and maintainability. Web development usage[edit] In the web development field, three-tier is often used to refer to websites , commonly electronic commerce websites, which are built using three tiers: A front-end web server serving static content, and potentially some cached dynamic content. In web-based application, front end is the content rendered by the browser. The content may be static or generated dynamically. A middle dynamic content processing and generation level application server e. A back-end database or data store , comprising both data sets and the database management system software that manages and provides access to the data. Other considerations[edit] Data transfer between tiers is part of the architecture. Often middleware is used to connect the separate tiers. Separate tiers often but not necessarily run on separate physical servers, and each tier may itself run on a cluster. Traceability[edit] The end-to-end traceability of data flows through n-tier systems is a challenging task which becomes more important when systems increase in complexity. The Application Response Measurement defines concepts and APIs for measuring performance and correlating transactions between tiers. Generally, the term "tiers" is used to describe physical distribution of components of a system on separate servers, computers, or networks processing nodes. A three-tier architecture then will have three processing nodes. The term "layers" refer to a logical grouping of components which may or may not be physically located on one processing node.

Chapter 8 : Approaches to Building Client/Server Applications

Client-server architecture, architecture of a computer network in which many clients (remote processors) request and receive service from a centralized server (host computer). Client computers provide an interface to allow a computer user to request services of the server and to display the results the server returns.

Here are a few other attributes of server-side code: Is never seen by the user except within a rare malfunction Stores data such as user profiles, tweets, pages, etcâ€¦ Creates the page the user requested With client-side code, languages used include: Moreover, client-side code can be seen and edited by the user. Plus, it has to communicate only through HTTP requests and cannot read files off of a server directly. Furthermore, it reacts to user input. Web Application Architecture is Important for Supporting Future Growth The reason why it is imperative to have good web application architecture is because it is the blueprint for supporting future growth which may come from increased demand, future interoperability and enhanced reliability requirements. Through object-oriented programming , the organizational design of web application architecture defines precisely how an application will function. Delivering persistent data through HTTP, which can be understood by client-side code and vice-versa Making sure requests contain valid data Offers authentication for users Limits what users can see based on permissions Creates, updates and deletes records Trends in Web Application Architecture As technology continues to evolve, so does web application architecture. One such trend is the use of and creation of service-oriented architecture. This is where most of the code for the entire application exists as services. As a result, one facet of the code can make a request to another part of the codeâ€"which may be running on a different server. Another trend is a single-page application. This is where web UI is presented through a rich JavaScript application. In terms of requests, it uses AJAX or WebSockets for performing asynchronous or synchronous requests to the web server without having to load the page. The user then gets a more natural experience with limited page load interruptions. At their core, many web applications are built around objects. The objects are stored in tables via an SQL database. Each row in a table has a particular record. So, with relational databases, it is all about relations. You can call on records just by listing the row and column for a target data point. With the two above trends, web apps are now much better suited for viewing on multiple platforms and multiple devices. Even when most of the code for the apps remain the same, they can still be viewed clearly and easily on a smaller screen. Best Practices for Good Web Application Architecture You may have a working app, but it also needs to have good web architecture. Here are several attributes necessary for good web application architecture: Not to mention, by supporting horizontal and vertical growth, software deployment is much more efficient, user-friendly and reliable. Additional Resources and Tutorials on Web Application Architecture To learn more about best practices for sound web application architecture, including some helpful tutorials, visit the following resources:

Chapter 9 : Application Architecture

Introduction to Client/Server Architecture. In the Oracle database system environment, the database application and the database are separated into two parts: a front-end or client portion, and a back-end or server portionâ€"hence the term client/server architecture.

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. 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 web server and file server software at the same time to serve different data to clients making different kinds of requests. Client software can also communicate with server software within the same computer. Client and server communication[edit] In general, 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. Clients and servers exchange messages in a requestâ€"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 client-server 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. By restricting communication to a specific content format , it facilitates parsing. By abstracting access, it facilitates cross-platform data exchange. 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 , server software may limit the availability to clients. Finally, the web server returns the result to the client web browser for display. In each step of this sequence of clientâ€"server message exchanges, a computer processes a request and returns data. This is the request-response messaging pattern. When all the requests are met, the sequence is complete and the web browser presents the data to the customer. One context in which researchers used these terms was in the design of a computer network programming language called Decode-Encode Language DEL. 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 clientâ€"server transaction. 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 user-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 clientâ€"server model, a server is more likely to be devoted to the task of serving. History of personal computers , Decentralized computing , and Computer cluster The clientâ€"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 amount 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. In contrast, a fat 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 s to the late s, many organizations transitioned computation from centralized servers, such as mainframes and minicomputers , to fat clients. 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 s. In the clientâ€"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 work-load i. Load-balancing and failover systems are often employed to scale the server implementation. Peers are coequal, or equipotent nodes in a non-hierarchical network. Unlike clients in a clientâ€”server or clientâ€”queueâ€”client network, peers communicate with each other directly. 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.