System Architectures

Deciding on software components, their **interaction**, and their **placement** leads to an instance of the software architecture, also known as system architecture.

Centralised Organisations

Clients are those that require services from the *server*

Simple client-server architecture

A **server** is a process implementing a specific service, for example, a file system service or a database service.

A **client** is a process that requests a service a server by sending it a request and subsequently waiting for the server's reply.

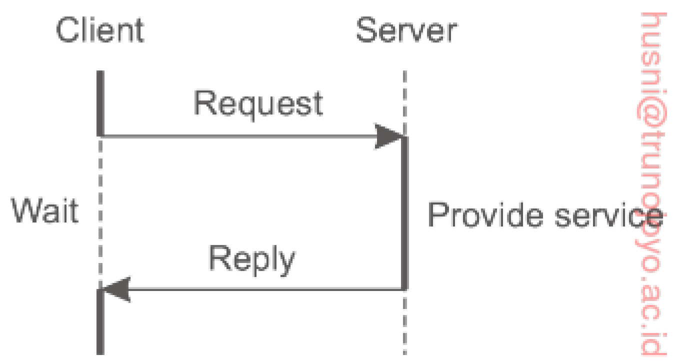


Figure 2.15: General interaction between a client and a server.

Communication between a server and a client can be implemented by a simple *connectionless* protocol where the underlying network is fairly reliable in many local-area networks.

- When a client requests a service, it simply packages a message for the server, identfying the service it wants, along with necessary input data.
- The message is then sent to the server, which is always waits for an incoming request, subsequently processes it, and package the results in a reply message that is sent to the client.

ADVANTAGES OF CONNECTIONLESS PROTOCOL

- Efficient
- As long as messages do not get lost or corrupted, it works fine.

DISADVANTAGES

- Susceptible to transmission failures, the best it can do is let the client resend the request when no reply messages comes in, the client then cannot know if the original message was lost or the reply transmission failed.

When an operation can be repeated multiple times without harm, it is said to **idempotent**. **No single solution for handling lost messages** is available since some requests can be idempotent and other might be not.

ALTERNATIVE: CONNECTION ORIENTED PROTOCOL

- Low performance, not suitable for LANs but opposite is true for WANs.
- Example: TCP/IP connections, performance cost for establishing and tearing down connections.

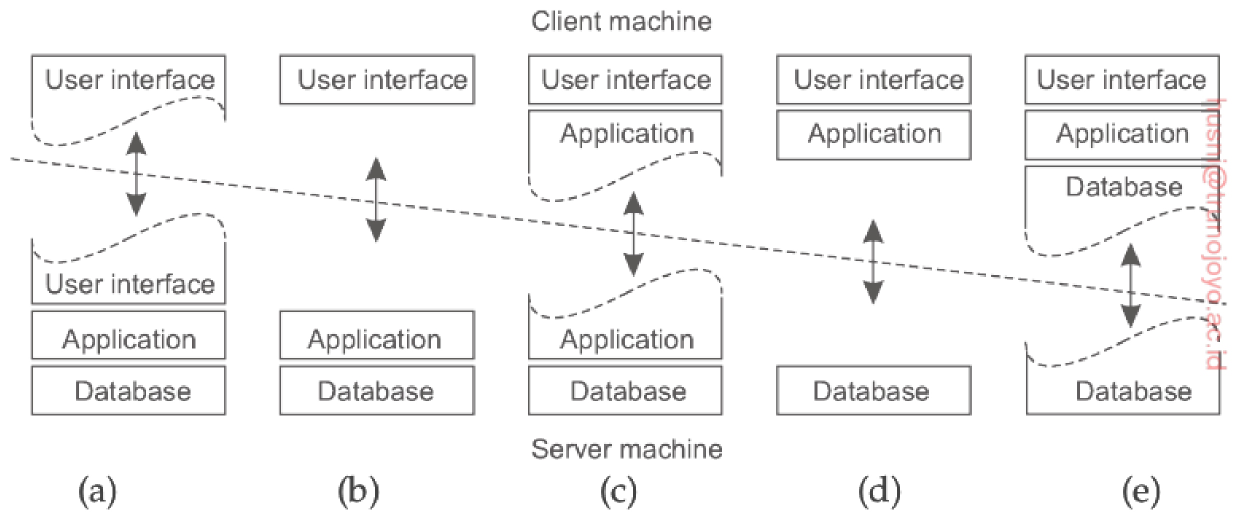
Multitiered architectures

In this organisation, everything is handled by the server, the client is nothing more than a dumb terminal, possibly with only a convenient graphical interface.

Many distributed system are divided into:

1. User interface layer
2. Processing Layer
3. Data Layer

One approach for organising clients and servers is to distribute these layers across different machines. Making a distinction between client and server machine is referred to as **two-tiered architecture**.



EXPLANATION

- In a), only the terminal dependent part is on the client side
- In b), the entire user interface is on the client side
- In c), part of the application is on the client side, example: doing some processing on form input before sending a request to the server

- In d), the processing is done on the client side but all the files or database entries goes to the server. Example: Banking applications
- In e), part of the data is on the client's local hard disk, example: keeping cached data on the client machine.

The **server can sometimes act as a client**, leading to a **three-tiered architecture**.

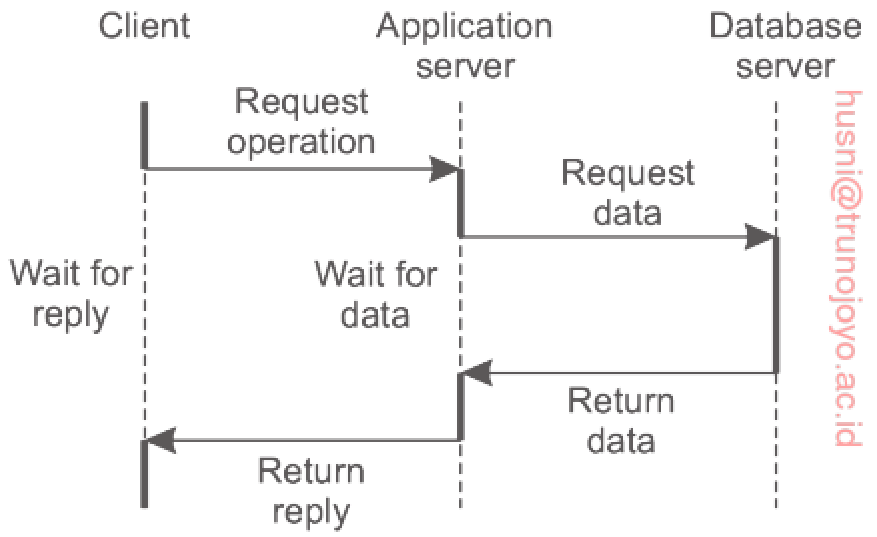


Figure 2.17: An example of a server acting as client.

A Web server acts as an entry point to a site, passing requests to an **application server** where the actual processing takes place. This application server, in turn, interacts with a database server. For example, an application server may be responsible for running the code to inspect the available inventory of some goods as offered by an electronic bookstore. To do so, it may need to interact with a database containing the raw inventory data.

Decentralised Architectures

In many business environments, distributed processing is equivalent to organizing a client-server application as a multitiered architecture . This type of distribution is referred to as **vertical distribution**. The characteristic feature of vertical distributions is that it is achieved by logically placing different components on different machines.

In **horizontal distribution**, a client or server may be physically split into logically split into equivalent parts, but each part is operating on its own share of the complete dataset, thus balancing the load.

Peer-to-Peer systems (P2P)

From a higher-level perspective, the processes that constitute the peer-to-peer system are all equal. The functions that need to be to carried are represented by every process that constitute the distributed system.

Much of the interaction between the processes are symmetric : each process will act as a client and a server at the same time.

Overlay Network: A network in which a nodes formed by the process and the links represent the possible communication channels.

Two types of overlay networks:

1. Structured
2. Unstructured

Structured P2P Networks

The nodes are organised in an overlay adheres to a specific, deterministic topology: a ring, a binary tree, a grid, etc. Generally based on using a so called *semantic-free index* (each data item to be maintained by the system is uniquely associated with a key and this key = index).

`key(data item) = hash(data item's value)`

`(key, value)` pairs are stored in this P2P networks in a **distributed hash table (DHT)**.

Mapping a key to an existing node:

`existing node = lookup(key)`

Any node can be asked to lookup a given key, which then boils down to efficiently *routing* that lookup request to the node responsible for storing the data associated with that key.

Unstructured P2P Networks

Each Node maintains an adhoc list of neighbors. The Resulting overlay resembles what is called a **random graph**, in which an edge $< u, v >$ between two nodes u and v exists only with a certain possibility $\mathcal{P}[< u, v >]$ Ideally, this probability is same for all pairs of nodes, but not on practice.

In this, when a node joins the network, it often contacts a *well known node* to obtain a starting a list of other peers in the system. This list can be used to find more peers and ignore others and so on. In practice, the list changes continously, for example when a neighbor is no longer responsive it needs to be replaced.

FLOODING

- An issuing node u simply passes a request for a data item to all its neighbors.
- A request may be ignored, if its recieving node, say v, has seen it before. If v has the data, it can send it directly to u or send it back to the original *forwarder*, who will then return it to its forwarder and so on. If v does not have the data, it forwards the request to all its neighbors.
- Flooding can be expensive, each request has an associated **time-to-live** or **TTL**. Choosing the right TTL value is crucial.

RANDOM WALKS

An issuing node u can simply try to find a data item by asking a randomly chosen neighbor , say v.
If v does not have the data, it forwards the request to one of its randomly chosen neighbors, and so on.
The result is known as **random walk**.
Much less network traffic, but takes a lot of time to reach the node that has the requested data.

To decrease the waiting time, an issuer can simply start N random walks simultanously. Effective for small values of N such as 16 or 64.

A random walk also needs to be stopped. To this end, we can either again use a TTL, or alternatively, when a node receives a lookup request, check with the issuer whether forwarding the request to another randomly selected neighbor is still needed.

Hierarchically Organised Peer-to-Peer Networks

In unstructured p2p networks, it becomes problematic to loacte relevant data item as the size of the network grows because there is no deterministic way of routing of a lookup request to specific data item (random walks and flooding).

Alternative: Use of special nodes that maintain an index of data items.

In CDNs, the nodes may offer storage for hosting copies of web documents allowing web clients to access web pages nearby, and thus to access them quickly.

Need: a means to find out where the documents can be stored best => making use of a **broker** which collects data on resource usage and availability of number of nodes that are in each other's proximity will allow to quickly select a node with **sufficient resources**.

Super Peers: Nodes maintaining an index or acting as a broker.
Weak Peers: Regular peers.

Association between weak peer and super peer is fixed, when a weak peer joins a network, it attaches to one of the super peers and remains attached until it leaves the network.

Backup schemes: pairing every super peer with another one and requiring week peer to attach to both.

Fixed association may not always be the best solution. For example, in file sharing networks, it may be better for weak peer to connect to super peer maintaining an *index of files it is currently interested in*.

Leader Election Problem: How to select nodes eligible to become super peers.

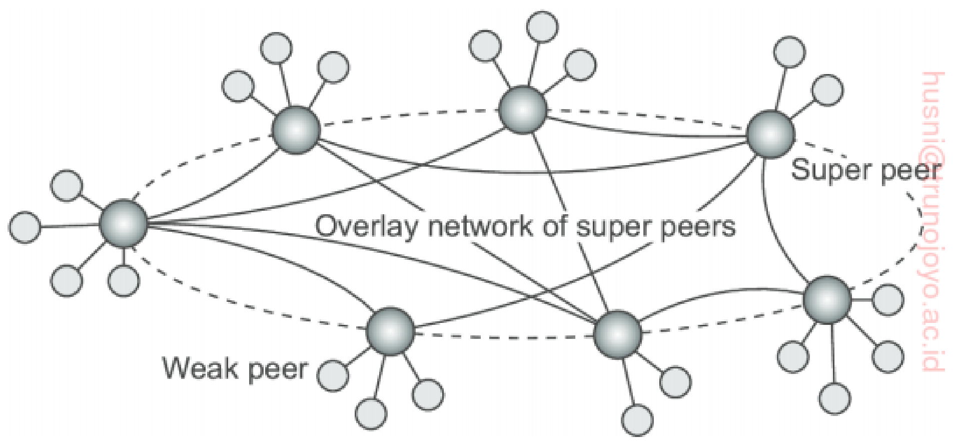


Figure 2.20: A hierarchical organization of nodes into a super-peer network.

Hybrid Architectures

Client-Server + Decentralised

Edge Server Systems

These systems are deployed on the Internet where **servers are "at the edge" of the network**. This edge is formed by the boundary between enterprise networks and the actual Internet, for example, as provided by the ISP. Likewise, where end users at home connect to the internet through their ISP, the ISP can be considered as residing at the edge of the internet.

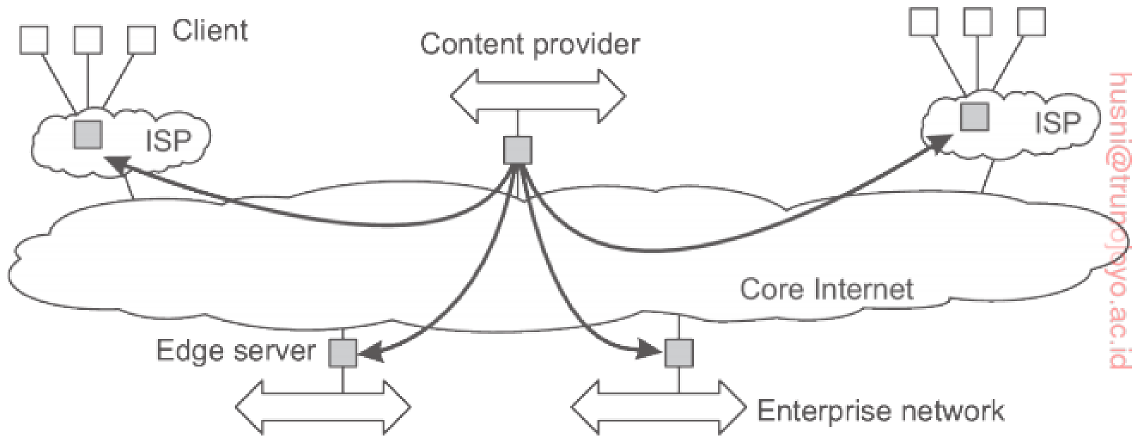


Figure 2.21: Viewing the Internet as consisting of a collection of edge servers.

End users, or clients in general connect to the Internet by means of an edge server. The edge server's main purpose is to deliver content, possibly after filtering and transcoding functions . A collection of edge servers can be used to optimise content and application distribution. The basic model is that for a specific organisation, one edge server acts as an origin server from which all contents originates .

Today: Cloud computing is implemented in a data center as the core, additional servers at the edge of the networks are used to assist in computtaions and storage, essentially leading to distributed cloud systems.

Collaborative Distributed Systems

In **BitTorrent**, when an end user is looking for a file, he downloads chunks of the file from other users until the downloaded chunks can be assembled together to yield the complete file. An important design goal was to ensure **collaboration**.

Free Riding: In most file sharing systems, a significant fraction of the participants merely download files but otherwise contribute close to nothing.

To prevent free riding in bittorent, a file can be downloaded only when the downloader is providing content to someone else.

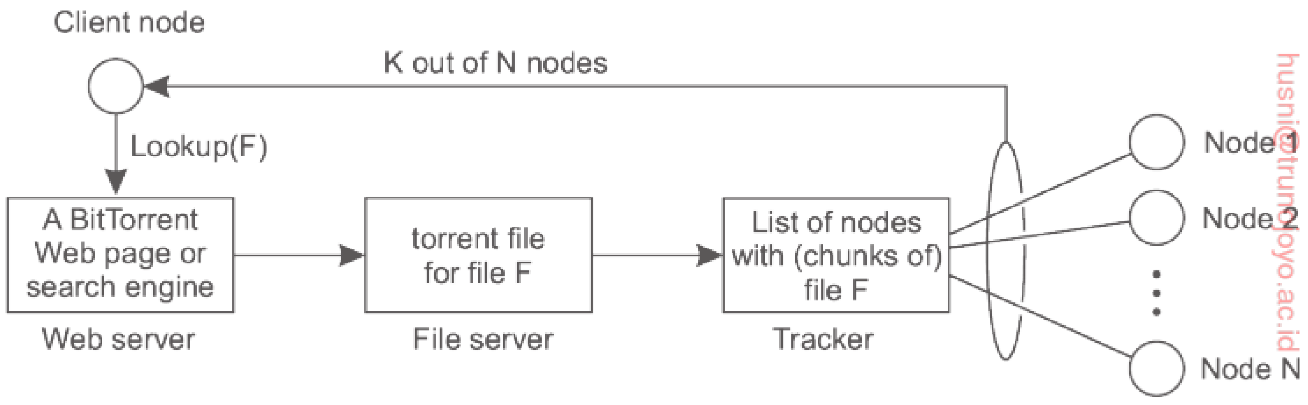


Figure 2.22: The principal working of BitTorrent [adapted with permission from Pouwelse et al. [2005]].

To download a file:

- A user needs to access a global directory, which is generally one of a few well-known Web sites. Such a directory contains references to what is known as torrent files.
- A **Torrent file** contains the information needed to download a specific file. In particular, it contains a link to what is known as a **tracker**, which is a server that is keeping an accurate account of *active* nodes that have the chunks of the files. One tracker per file.
- Once the nodes have been identified from where the chunks can be downloaded, the downloading node effectively becomes active. At that point, it is forced to help others.
- if node P notices that node Q is downloading more than it is uploading, P can decrease the rate at which it sends data to Q. This schemes works well provided that P has something to download from Q. For this reason, nodes are provided with references to other nodes putting them in better position to trade data.

Bottleneck: Trackers.

In an alternative implementation, a node also joins a seperate structured p2p network (i.e., **DHT**) to assist in tracking file downloads.

It is centralised + decentralised:

Centralised part = global directory.
De-centralised part = Nodes that have the chunks of the file + p2p network that a node joins i.e., DHT. (DHT = Distributed Hash Table).

The intial tracker for the requested file is looked up in the DHT through a so-called **magnet link**.

Example Architectures

The Network File System (NFS)

- Organised along the lines of client server architectures
- Belongs to sun Microsystems
- For unix systems
- NFSv3* and NFSv4

Basic Idea

Each file server provides a standardized view of its local file system. In other words, it should not matter how that local file system is implemented; each NFS server supports the same model.

NFS comes with communication protocol that allows the clients to access the files stored on a server, thus allowing a heterogeneous collection of processes, possibly running on different OS and machines, to **share a common file system**.

Model

Model: **Remote File service**

Clients are offered transparent access to a file system that is managed by a remote server. However, clients are normally unaware of the actual location of the files . Instead, they are offered interface to a file system that is similar to the interface offered by a conventional local file system.

In particular, the client is offered only an interface containing various file operations, but the server is responsible for implementing those operations.
This model is referred to as **remote access model**.

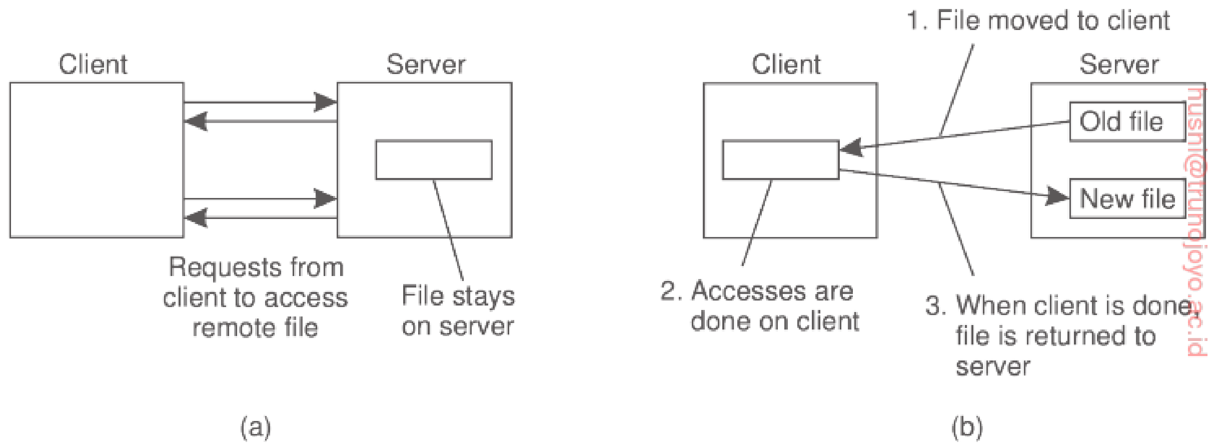

Client
Server
Requests from client to access remote file
File stays on server
(a)
1. File moved to client
Client
Server
Old file
New file
2. Accesses are done on client
3. When client is done, file is returned to server
(b)

Figure 2.24: (a) The remote access model. (b) The upload/download model.

In contrast, in the **upload/download model** a client accesses a file locally after having it downloading it from the server. When the client is finished with the file, it is uploaded back to the server again so that it can be used by another client. Example: FTP service.

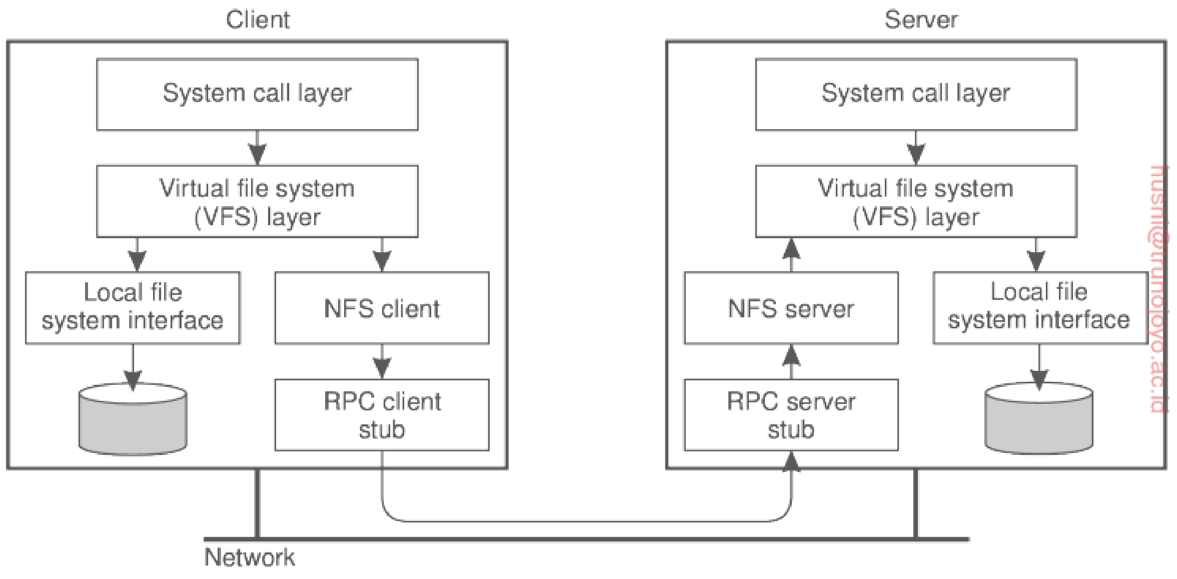

Client
System call layer
Virtual file system (VFS) layer
Local file system interface
NFS client
RPC client stub
Server
System call layer
Virtual file system (VFS) layer
NFS server
Local file system interface
RPC server stub
Network

Figure 2.25: The basic NFS architecture for Unix systems.

The Local Unix file system interface is replaced by an interface to Virtual File system (VFS) which is a now de facto standard for interfacing to distributed file systems.

With NFS, operations on the VFS interface are either passed to a local file system, or passed to seperate component known as **NFS Client**, which takes care of handling access to files stored at a remote server.

In NFS, all client-server communication is done through so called remote procedure calls (**RPC**s).

A RPC is standardized way to let a client on machine A make an ordinary call to a procedure that is implemented on another machine B.

Operations offered by VFS interface can be different from those offered by NFS client. The whole idea of VFS is to hide the difference between various file systems.

On the server side, the NFS server is responsible for handling incoming client requests. The RPC component at the server converts the incoming requests to regular VFS file operations that are subsequently passed to the VFS layer. Again, the VFS is reponsible for implementing a local file system in which the actual files are stored.

Important Advantage: NFS is independent of local file systems. Does not matter which OS the client or server is using.

The Web

Simple Web-based Systems

Organised as relatively simple client-server architectures. The core of a Web site is formed by a process that has access to a local file system storing web documents.

The simplest way to refer to a document is by means of a reference called a **Uniform Resource Locator (URL)**. It specifies where a document is located by emebedding the DNS name of its associated server along with a filename by which the server can lookup the document in its local file system. Furthermore, a URL spcifies the application-level protocol for transferring the document across the network.

A client interacts with web server through a **browser**, which is responsible for properly displaying a document.

The communication between a client and a web server is standardized: they both adhere to the **HyperText Transfer Protocol (HTTP)**.

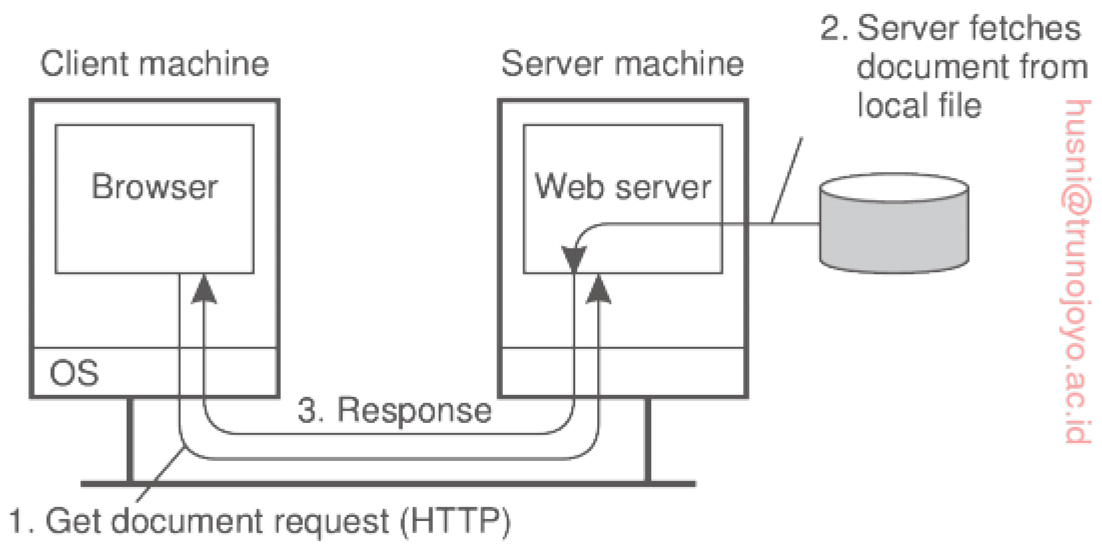

Client machine
Browser
OS
Server machine
Web server
2. Server fetches document from local file
3. Response
1. Get document request (HTTP)

Figure 2.27: The overall organization of a traditional Web site.

HTML doc contains various instructions expressing how its contents should be displayed. Documents can contain more than just markup instructions, like **Javascript**.

Multitiered Architectures

Documents have become completely dynamic. Most things are generated "on the spot" as a result of sending a request to the server along with client-side scripts and such to composed on-the-fly

Support for simple user interaction by means of **Common gateway interface (CGI)**

CGI defines a standard way by which a web server can execute a program taking user data as input. Usually the data comes from an HTML form; it specifies the program to be executed at the server side, along with the parameter values that are filled in by the user.

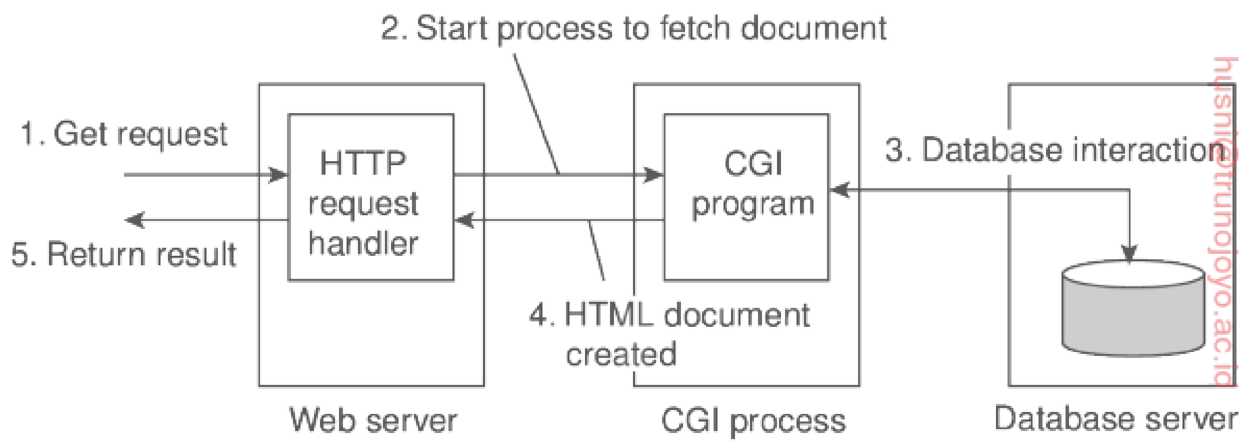


Figure 2.28: The principle of using server-side CGI programs.

CGI programs can be as sophisticated as the developer wants.

Server nowadays do much more than just fetching documents. One of the most important enhancements is that servers can also process a document before passing it to the client, that is it may contain **server-side scripts**, which is executed by the server when the documents has been fetched locally.