Date:31.01.2022

Third Year B. Tech., Sem VI 2021-22

Advanced Database System Lab

Assignment submission

PRN No: 2019BTECS00064

Full name: Kunal Santosh Kadam

Batch: T2

Assignment: 1

Title of assignment: Mini-seminar/ Presentation/ Prototype-demo on topic assigned to each

Topic: Distributed Web-Based System

Introduction

The World Wide Web (WWW) can be viewed as a huge distributed system consisting of millions of clients and servers for accessing linked documents. Servers maintain collections of documents, while clients provide users an easy-to-use interface for presenting and accessing those documents.

The Web started as a project at CERN, the European Particle Physics Laboratory in Geneva, to let its large and geographically dispersed group of researchers access shared documents using a simple hypertext system. A document could be anything that could be displayed on a user's computer terminal, such as personal notes, reports, figures, blueprints, drawings, and so on. By linking documents to each other, it became easy to integrate documents from different projects into a new document without the necessity for

centralized changes. The only thing needed was to construct a document providing links to other relevant documents [see also Berners-Lee et al. (1994)].

Since 1994, Web developments have been initiated by the World Wide Web Consortium, a collaboration between CERN and M.I.T. This consortium is responsible for standardizing protocols, improving interoperability, and further enhancing the capabilities of the Web. In addition, we see many new developments take place outside this consortium, not always leading to the capability one would hope for. By now, the Web is more than just a simple document-based system. Notably with the introduction of Web services we are seeing a huge distributed system emerging in which services rather than just documents are being used, composed, and offered to any user or machine that can find use of them.

Architecture

The architecture of Web-based distributed systems is not fundamentally different from other distributed systems. However, it is interesting to see how the initial idea of supporting distributed documents has evolved since its inception in 1990s. Documents turned from being purely static and passive to dynamically generated containing all kinds of active elements. Furthermore, in recent years, many organizations have begun supporting services instead of just documents. In the following, we discuss the architectural impacts of these shifts.

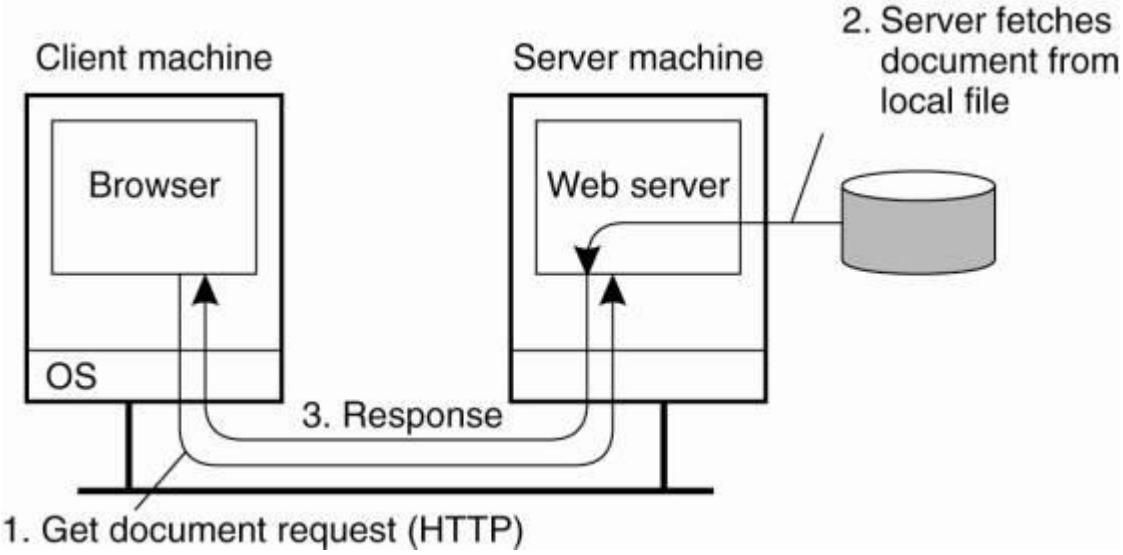
Traditional Web-Based Systems

Unlike many of the distributed systems we have been discussing so far, Web based distributed systems are relatively new. In this sense, it is somewhat difficult to talk about traditional Web-based systems, although there is a clear distinction between the systems that were available at the beginning and those that are used today.

Many Web-based systems are still organized 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 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, often by embedding the DNS name of

its associated server along with a file name by which the server can look up the document in its local file system. Furthermore, a URL specifies the application-level protocol for transferring the document across the network. There are several different protocols available, as we explain below.

A client interacts with Web servers through a special application known as a browser. A browser is responsible for properly displaying a document. Also, a browser accepts input from a user mostly by letting the user select a reference to another document, which it then subsequently fetches and displays. The communication between a browser and Web server is standardized: they both adhere to the HyperText Transfer Protocol (HTTP) which we will discuss below. This leads to the overall organization shown in figure.



The Web has evolved considerably since its introduction. By now, there is a wealth of methods and tools to produce information that can be processed by Web clients and Web servers. In the following, we will go into detail on how the Web acts as a distributed system. However, we skip most of the methods and tools used to construct Web documents, as they often have no direct relationship to the distributed nature of the Web. A good introduction on how to build Web-based applications can be found in Sebesta (2006).

Web Documents

Fundamental to the Web is that virtually all information comes in the form of a document. The concept of a document is to be taken in its broadest sense: not only can it contain plain text, but a document may also include all

kinds of dynamic features such as audio, video, animations and so on. In many cases, special helper applications are needed to make a document "come to life." Such interpreters will typically be integrated with a user's browser.

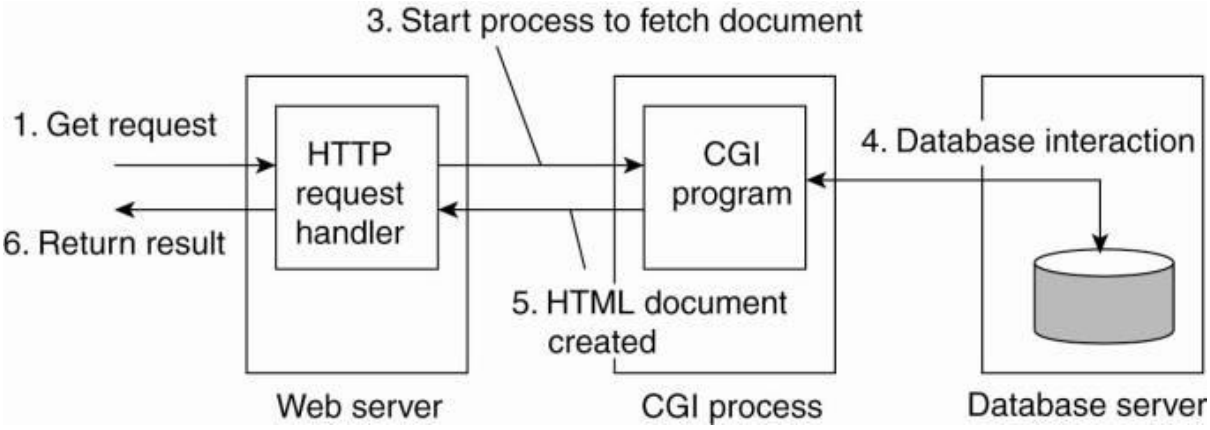
HTML and XML can include all kinds of tags that refer to embedded documents, that is, references to files that should be included to make a document complete. It can be argued that the embedded documents turn a Web document into something active. Especially when considering that an embedded document can be a complete program that is executed on-the-fly as part of displaying information, it is not hard to imagine the kind of things that can be done.

Type	Subtype	Description
Text	Plain	Unformatted text
	HTML	Text including HTML markup commands
	XML	Text including XML markup commands
Image	GIF	Still image in GIF format
	JPEG	Still image in JPEG format
Audio	Basic	Audio, 8-bit PCM sampled at 8000 Hz
	Tone	A specific audible tone
Video	MPEG	Movie in MPEG format
	Pointer	Representation of a pointer device for presentations
Application	Octet-stream	An uninterpreted byte sequence
	Postscript	A printable document in Postscript
	PDF	A printable document in PDF
Multipart	Mixed	Independent parts in the specified order
	Parallel	Parts must be viewed simultaneously

Multitiered Architecture

The combination of HTML (or any other markup language such as XML) with scripting provides a powerful means for expressing documents. However, we have hardly discussed where documents are actually processed, and what kind of processing takes place. The WWW started out as the relatively simple two-tiered client-server system shown previously in Fig. 12-1. By now, this simple architecture has been extended with numerous components to support the advanced type of documents we just described.

One of the first enhancements to the basic architecture was support for simple user interaction by means of the Common Gateway Interface or simply CGI. CGI defines a standard way by which a Web server can execute a program taking user data as input. Usually, user data come from an HTML form; it specifies the program that is to be executed at the server side, along with parameter values that are filled in by the user. Once the form has been completed, the program's name and collected parameter values are sent to the server, as shown in Figure.



When the server sees the request, it starts the program named in the request and passes it the parameter values. At that point, the program simply does its work and generally returns the results in the form of a document that is sent back to the user's browser to be displayed.

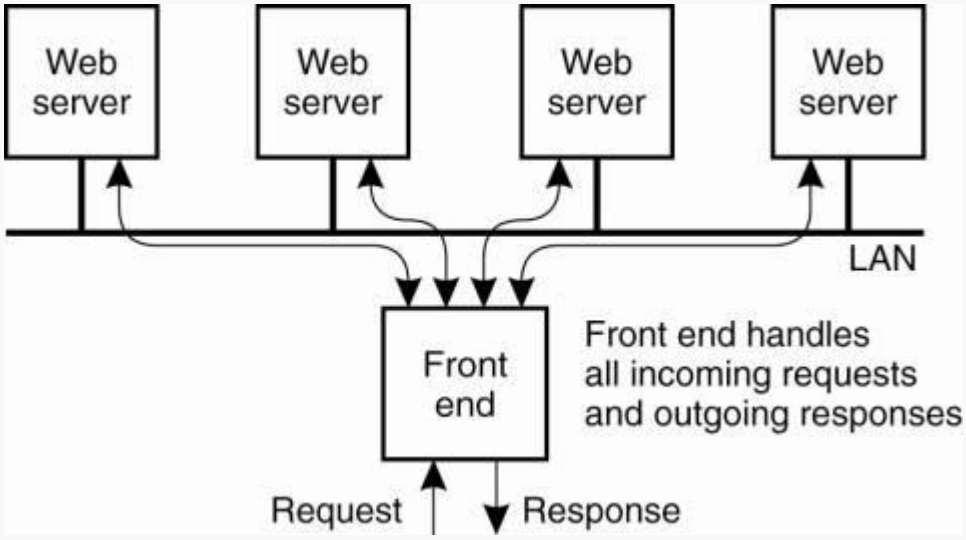
CGI programs can be as sophisticated as a developer wants. For example, as shown in Fig. many programs operate on a database local to the Web server. After processing the data, the program generates an HTML document and returns that document to the server. The server will then pass the document to the client. An interesting observation is that to the server, it appears as if it is asking the CGI program to fetch a document. In other words, the server does nothing but delegate the fetching of a document to an external program.

The main task of a server used to be handling client requests by simply fetching documents. With CGI programs, fetching a document could be delegated in such a way that the server would remain unaware of whether a document had been generated on the fly, or actually read from the local file system. Note that we have just described a two-tiered organization of server-side software.

This three-tiered organization introduces a problem, however: a decrease in performance. Although from an architectural point of view it makes sense to distinguish three tiers, practice shows that the application server and database are potential bottlenecks. Notably improving database performance can turn out to be a nasty problem. We will return to this issue below when discussing caching and replication as solutions to performance problems.

Web Server Clusters

An important problem related to the client-server nature of the Web is that a Web server can easily become overloaded. A practical solution employed in many designs is to simply replicate a server on a cluster of servers and use a separate mechanism, such as a front end, to redirect client requests to one of the replicas. This principle is shown in Figure, and is an example of horizontal distribution.



A crucial aspect of this organization is the design of the front end. as it can become a serious performance bottleneck, what will all the traffic passing through it. In general, a distinction is made between front ends operating as transportlayer switches, and those that operate at the level of the application layer.

Whenever a client issues an HTTP request, it sets up a TCP connection to the server. A transport-layer switch simply passes the data sent along the TCP connection to one of the servers, depending on some measurement of the server's load. The response from that server is returned to the switch, which will then forward it to the requesting client. As an optimization, the switch and

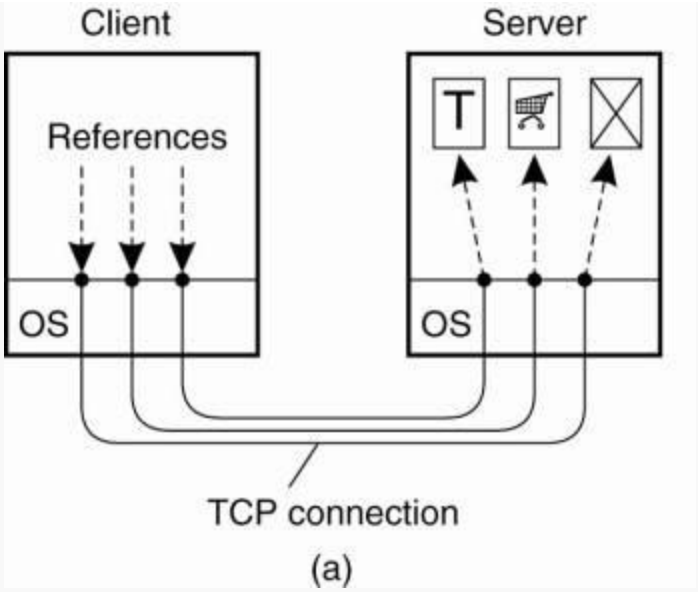
servers can collaborate in implementing a TCP handoff. The main drawback of a transport-layer switch is that the switch cannot take into account the content of the HTTP request that is sent along the TCP connection. At best, it can only base its redirection decisions on server loads.

HTTP

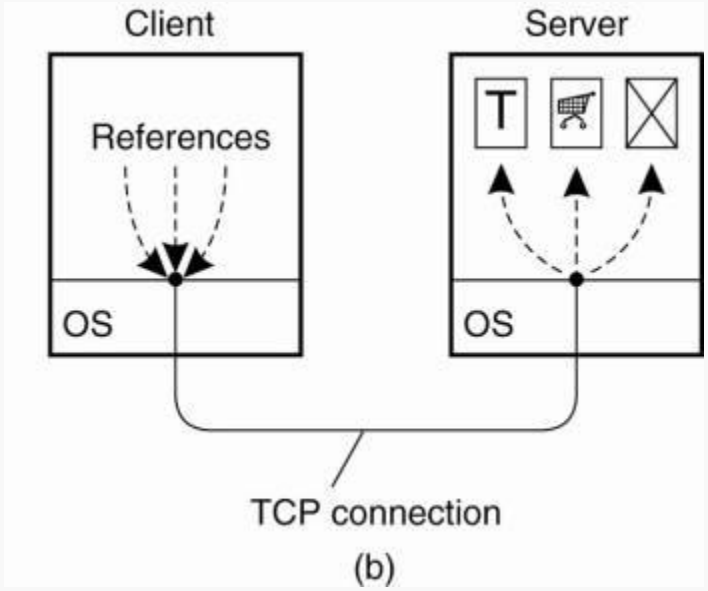
All communication in the Web between clients and servers is based on the Hypertext Transfer Protocol (HTTP). HTTP is a relatively simple client-server protocol; a client sends a request message to a server and waits for a response message. An important property of HTTP is that it is stateless. In other words, it does not have any concept of open connection and does not require a server to maintain information on its clients. HTTP is described in Fielding et al. (1999).

HTTP is based on TCP. Whenever a client issues a request to a server, it first sets up a TCP connection to the server and then sends its request message on that connection. The same connection is used for receiving the response. By using TCP as its underlying protocol, HTTP need not be concerned about lost requests and responses.

In HTTP version 1.0 and older, each request to a server required setting up a separate connection, as shown in Figure. When the server had responded, the connection was broken down again. Such connections are referred to as being nonpersistent. A major drawback of nonpersistent connections is that it is relatively costly to set up a TCP connection. As a consequence, the time it can take to transfer an entire document with all its elements to a client may be considerable.



(a)



(b)

Another approach that is followed in HTTP version 1.1 is to make use of a persistent connection, which can be used to issue several requests (and their respective responses), without the need for a separate connection for each (request, response)-pair. To further improve performance, a client can issue several requests in a row without waiting for the response to the first request.

Web Services

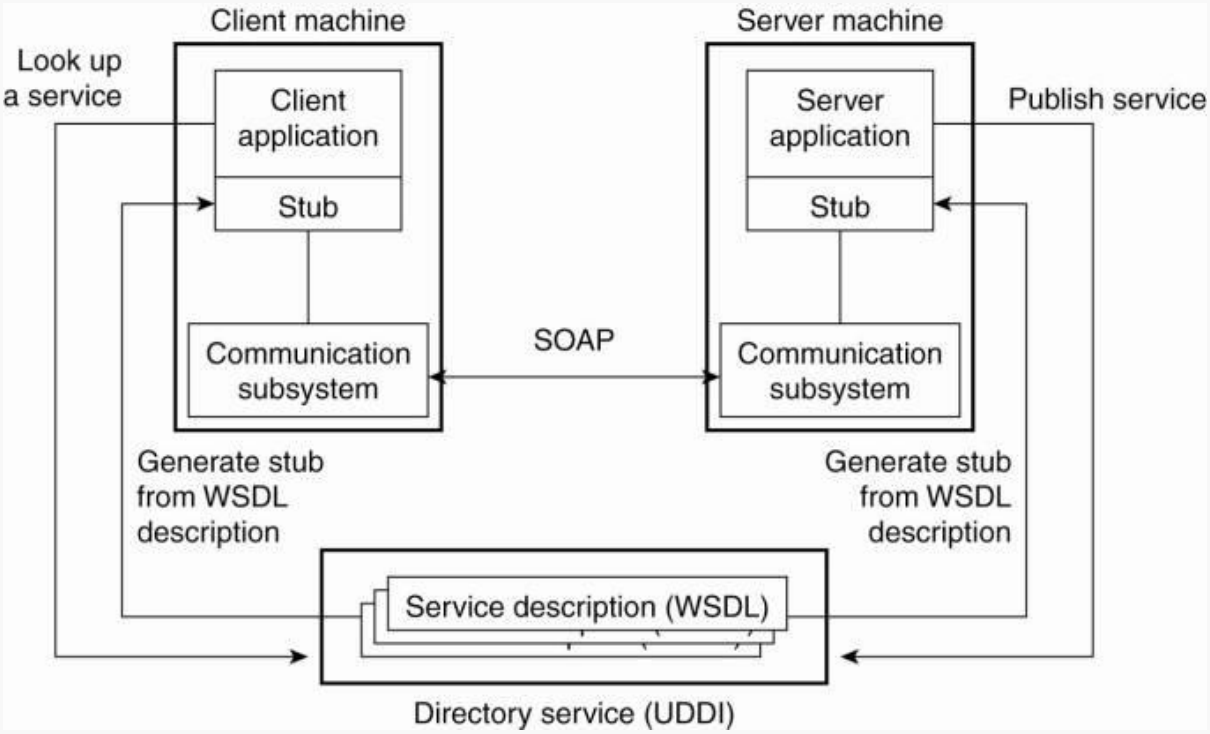
So far, we have implicitly assumed that the client-side software of a Web-based system consists of a browser that acts as the interface to a user. This assumption is no longer universally true anymore. There is a rapidly growing group of Web-based systems that are offering general services to

remote applications without immediate interactions from end users. This organization leads to the concept of Web services (Alonso et al., 2004).

Web Services Fundamentals

Simply stated, a Web service is nothing but a traditional service (e.g., a naming service, a weather-reporting service, an electronic supplier, etc.) that is made available over the Internet. What makes a Web service special is that it adheres to a collection of standards that will allow it to be discovered and accessed over the Internet by client applications that follow those standards as well. It should come as no surprise then, that those standards form the core of Web services architecture.

The principle of providing and using a Web service is quite simple, and is shown in Figure. The basic idea is that some client application can call upon the services as provided by a server application. Standardization takes place with respect to how those services are described such that they can be looked up by a client application. In addition, we need to ensure that service call proceeds along the rules set by the server application. Note that this principle is no different from what is needed to realize a remote procedure call.



An important component in the Web services architecture is formed by a directory service storing service descriptions. This service adheres to the

Universal Description, Discovery and Integration standard (UDDI). As its name suggests, UDDI prescribes the layout of a database containing service descriptions that will allow Web service clients to browse for relevant services.

Services are described by means of the Web Services Definition Language (WSDL) which is a formal language very much the same as the interface definition languages used to support RPC-based communication. A WSDL description contains the precise definitions of the interfaces provided by a service, that is, procedure specification, data types, the (logical) location of services, etc. An important issue of a WSDL description is that can be automatically translated to client-side and server-side stubs, again, analogous to the generation of stubs in ordinary RPC-based systems.

Finally, a core element of a Web service is the specification of how communication takes place. To this end, the Simple Object Access Protocol (SOAP) is used, which is essentially a framework in which much of the communication between two processes can be standardized. We will discuss SOAP in detail below, where it will also become clear that calling the framework simple is not really justified.