

#01 Client/Server Computing

CLIENT/SERVER COMPUTING AND WEB TECHNOLOGIES

Distinct characteristics of C/S

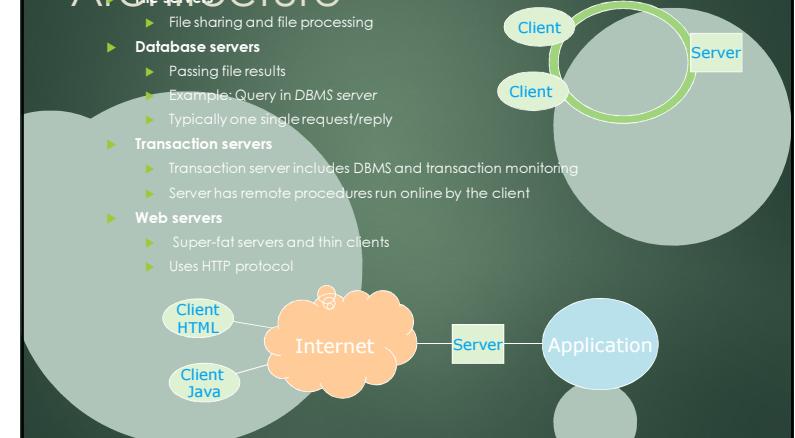
- ▶ Client-server is a computing architecture which separates a client from a server
- ▶ It is almost always implemented over a computer network
- ▶ The most basic type of client-server architecture employs only two types of nodes: clients and servers.
- ▶ This type of architecture is sometimes referred to as *two-tier*.
- ▶ It allows devices to share files and resources.
- ▶ Server provides the service
- ▶ Client is considered as the customer requesting the service

Distinct characteristics of C/S

- ▶ The server service can be shared among a number of clients
- ▶ Clients must request or initiate the service
- ▶ The location of the server in the network is transparent to clients
- ▶ Transaction between C/S is message-passing based
- ▶ C/S architecture is scalable
 - ▶ horizontally (more clients can added)
 - ▶ Vertically (more servers can be added)
- ▶ The server is centrally maintained where as clients are independent of each other

Systems with C/S Architecture

- ▶ **File servers**
 - ▶ File sharing and file processing
- ▶ **Database servers**
 - ▶ Passing file results
 - ▶ Example: Query in DBMS server
 - ▶ Typically one single request/reply
- ▶ **Transaction servers**
 - ▶ Transaction server includes DBMS and transaction monitoring
 - ▶ Server has remote procedures run online by the client
- ▶ **Web servers**
 - ▶ Super-fat servers and thin clients
 - ▶ Uses HTTP protocol



Client/Server Models

5

- ▶ Where to push the application to
- ▶ Fat clients
 - ▶ The bulk of the application is running on the client
 - ▶ The client knows how the data is organized and where it is
 - ▶ Different clients access the same applications different ways
- ▶ Fat servers
 - ▶ The server more complicated
 - ▶ The clients are less complex
 - ▶ More of the code runs on the server
 - ▶ The network interaction is minimized



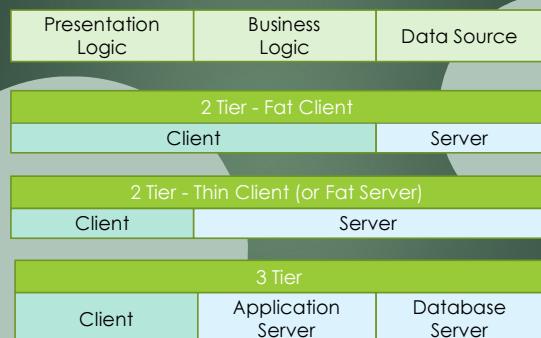
Two-Tier vs. Three-Tier

6

- ▶ Same basic idea as fat-client versus fat-server
- ▶ Depends on how the application is divided between the server and the client
- ▶ Two-tier servers
 - ▶ Examples: file servers and database server
 - ▶ In this case the process (application logic) is buried within the client or server (or both)
- ▶ Three-tier servers
 - ▶ Examples: Web and distributed objects
 - ▶ In this case the process is run on the middle-tier – separated from the user and data interface
 - ▶ They can integrate the data from multiple sources
 - ▶ More robust and more scalable

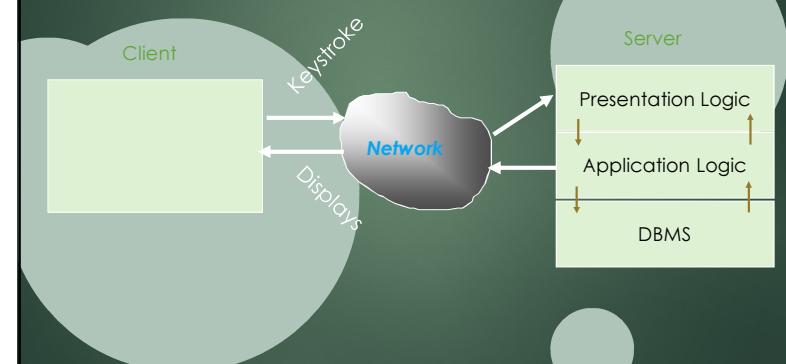
Tier Architecture

7



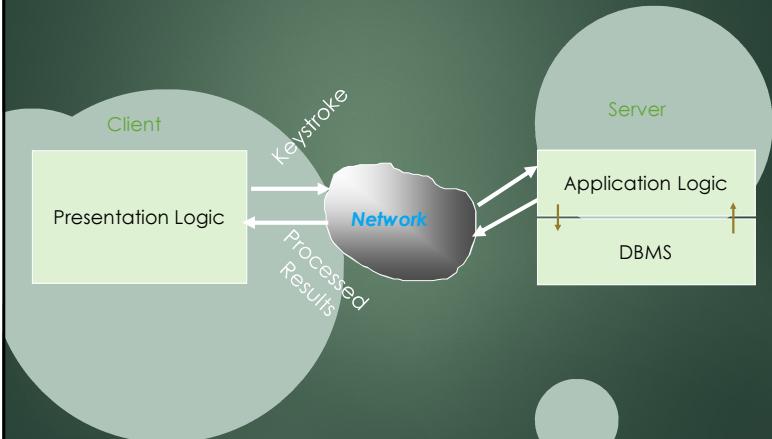
Client (dumb) - Server Model

8



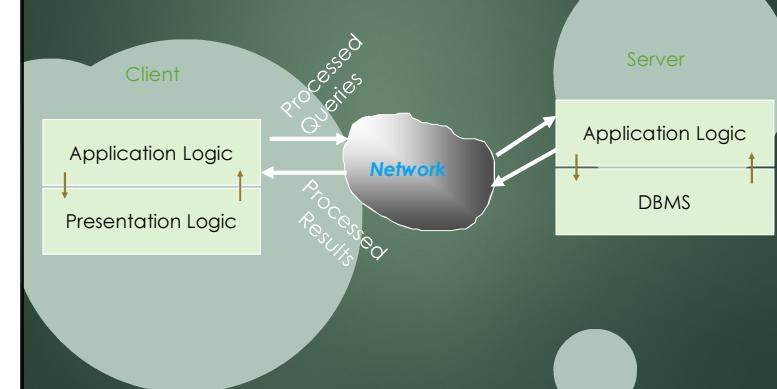
True Client-Server Model

9



Distributed Client-Server Model

10



Client/Server Computing

11

- ▶ Logical extension of modular programming
 - ▶ with assumption that separation of a huge program into modules can create
 - ▶ the possibility for further modification
 - ▶ easier development
 - ▶ better maintainability.
- ▶ All large modules need not all be executed within the same memory space.
 - ▶ the calling module becomes the client(requesting service)
 - ▶ the called module becomes the server (providing service).

Client/Server Computing

12

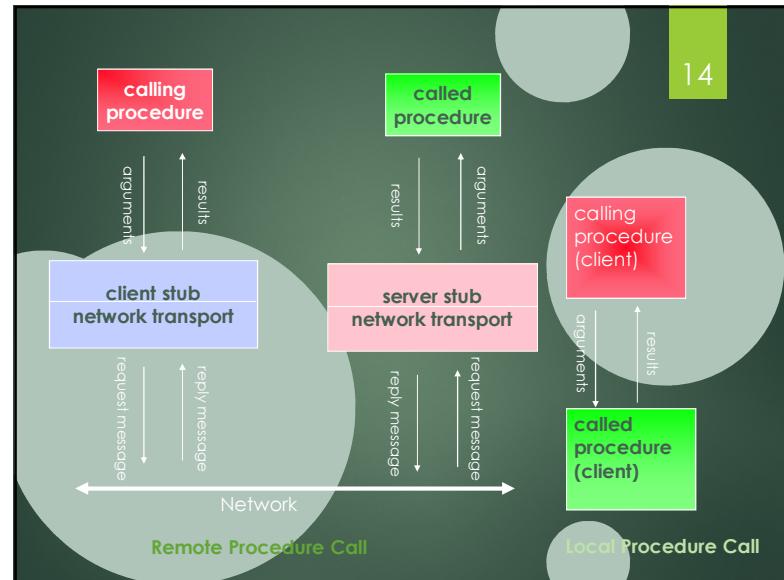
- ▶ Clients and Servers are running separately on appropriate hardware and software platforms for their functions.
 - ▶ For example, database management system servers running on platforms specially designed and configured to perform queries, or file servers running on platforms with special elements for managing files.
- ▶ Components in Client-Server Computing
 - ▶ Client
 - ▶ Server
 - ▶ Middleware

In client-server computing major focus is on SOFTWARE

Middleware Software

13

- ▶ It is the (/) between client and server which glues them together
 - ▶ Allowing the client request for a service and the server providing it
- ▶ Middleware can also be between server/server
- ▶ Two broad classes
 - ▶ General
 - ▶ LAN servers, TCP/IP, Communication stacks, Queuing services, etc.
 - ▶ Application specific
 - ▶ Used to accomplish a specific task
 - ▶ Groupware specific: SMTP
 - ▶ Internet specific: HTTP
 - ▶ Database specific: SQL



Six types of middleware

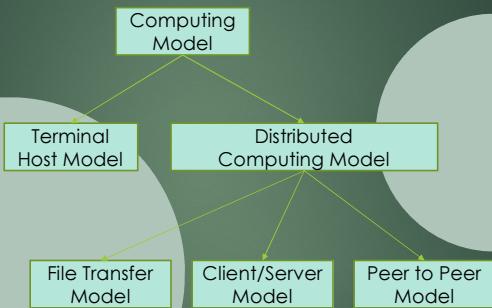
1. Asynchronous Remote Procedure Calls (RPC)
 - client makes calls to procedures running on remote computers but does not wait for a response
 - If connection is lost, client must re-establish the connection and send request again.
 - High scalability but low recovery, largely replaced by type 2
2. Synchronous RPC
 - distributed program may call services on different computers
 - makes it possible to achieve this without detailed coding (e.g. RMI in Java)
3. Publish/Subscribe (often called push technology)
 - server monitors activity and sends information to client when available.
 - It is asynchronous, the clients (subscribers) perform other activities between notifications from the server.
 - Useful for monitoring situations where actions need to be taken when particular events occur.

Six types of middleware

4. Message-Oriented Middleware (MOM)
 - asynchronous – sends messages that are collected and stored until they are acted upon, while the client continues with other processing.
5. Object Request Broker (ORB)
 - object-oriented management of communications between clients and servers.
 - ORB tracks the location of each object and routes requests to each object.
6. SQL-oriented Data Access
 - middleware between applications and database servers.
 - Has the capability to translate generic SQL into the SQL specific to the database

Computing Model

17



Peer-to-peer (P2P)

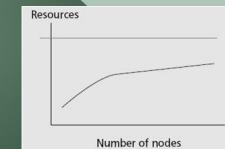
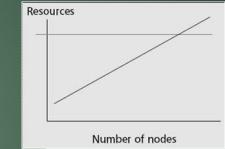
19/66

"Peer-to-peer is a way of structuring distributed applications such that the **individual nodes have symmetric roles**. Rather than being divided into clients and servers each with quite distinct roles, in P2P applications **a node may act as both a client and a server.**"

– Charter of Peer-to-peer Research Group, IETF/IETF, June 24, 2004
(<http://www.ietf.org/charters/p2prg.html>)

The architectures

- ▶ Server-based architecture
 - ▶ Client-Server / Server-Cluster
 - ▶ Problems :
 - ▶ Limited resources
 - ▶ All loads are centered on the server
 - ▶ Server-based architecture has low scalability.
 - ▶ The setup and maintenance cost is high.
- ▶ Peer-to-Peer (P2P) architecture
 - ▶ Advantages :
 - ▶ Distributing loads to all users
 - ▶ Users consume and provide resources
 - ▶ P2P architecture has high scalability.
 - ▶ The setup and maintenance cost is low.



Napster Sharing Style: hybrid center + edge

1. Users launch Napster and connect to Napster server
2. Napster creates dynamic directory from users' personal .mp3 libraries
3. *beastieboy* enters search criteria
4. Napster displays matches to *beastieboy*
5. *beastieboy* makes direct connection to *kingrook* for file transfer

"beastieboy"

- song1.mp3
- song2.mp3
- song3.mp3
- song4.mp3
- song5.mp3

"kingrook"

- song4.mp3
- song5.mp3
- song6.mp3
- song7.mp3

"slashdot"

- song5.mp3
- song6.mp3
- song7.mp3





- ▶ Farid Farahmand, "An Introduction to Client/Server Architecture"
- ▶ Rajkumar Buyya, "Client/Server Computing
(the wave of the future)"
- ▶ Albert Yau, "Client Server Computing",
http://www.doc.ic.ac.uk/~nd/surprise_95/journal/vol1/wcy/article1.html