INTRODUCTION

ARCHITECTURE OVERVIEW

Architectural Model

- An architectural model is concerned with the placement of its components and the relationships between them:
    - Client – Server Architecture
    - Peer - to - Peer Architecture
- The architecture of a system is its structure in terms of separately specified components and their interrelationships

4 Fundamental Building Blocks (And 4 Key Questions):

- Communicating entities: what are the entities that are communicating in the distributed architecture?
- Communication paradigms: how do these entities communicate, or, more specifically, what communication paradigm is used?
- Roles and responsibilities: what (potentially changing) roles and responsibilities do these entities have in the overall architecture?
- Placement: how are these entities mapped on to the physical distributed infrastructure (i.e., what is their placement)?

Architectural Models: Communicating Entities

System Perspective:

- communicating entities are processes
- distributed system: processes coupled with appropriate inter process communication paradigms

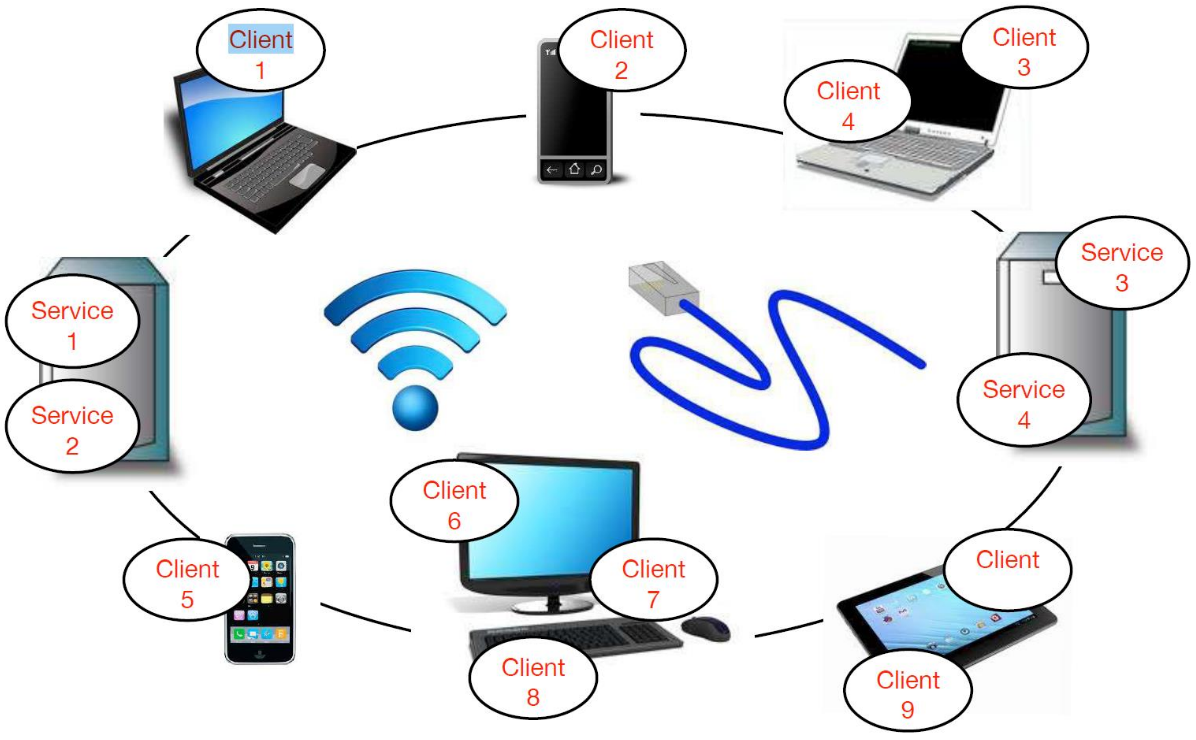


Figure 1 Processes vs Machine

Programming Perspective:

- More problem-oriented abstractions have been proposed, such as distributed objects, components, web services
- Distributed objects:
    - introduced to enable and encourage the use of object-oriented approaches in distributed systems
    - computation consists of a number of interacting objects representing natural units of decomposition for the given problem domain
    - objects are accessed via interfaces, with an associated interface definition language providing a specification of the methods defined on an object

Architectural Models : Communication Paradigms

//We are going to look at this in-depth in our subsequent lectures

– How do entities communicate in a distributed systems — Client - Server, Peer – to - Peer architectue? (What communication paradigm is used?)

3 Types of Communication Paradigm:

- Interprocess Communication
  low level support for communication between processes in the distributed system, including message-passing primitives, socket programming, multicast communication
- Remote Invocation
  most common communication paradigm, based on a two-way exchange between communicating entities and resulting in the calling of a remote operation (procedure or method)
- Indirect Communication
  communication is indirect, through a third entity, allowing a strong degree of decoupling between senders and receivers, in particular:
  - space uncoupling: senders do not need to know who they are sending to
  - time uncoupling: senders and receivers do not need to exist at the same time.
  - Key techniques include: group communication, publish subscribe systems, message queues, tuple spaces, distributed shared memory (DSM)

Architectural Models: Roles & Responsibilities

- What (potentially changing) roles and responsibilities do these entities have in the overall architecture?
- 2 architectural styles stemming from the role of individual processes

Client - Server Architecture

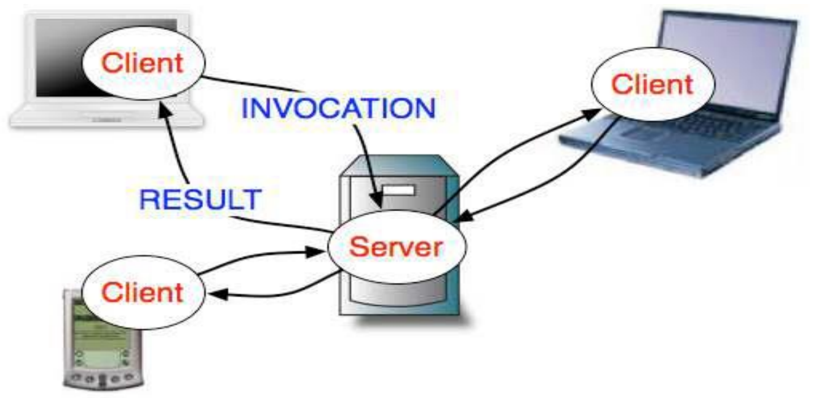


Figure 2 Client - Server Architecture

Peer - Peer Architecture

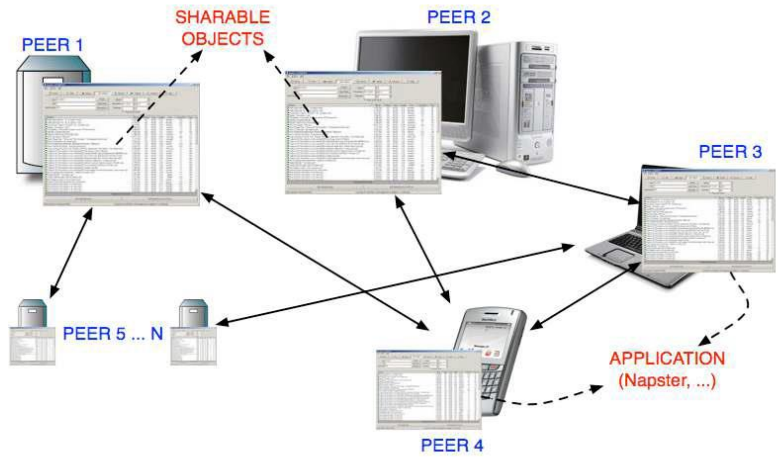


Figure 3 Peer - Peer Architecture

Client - Server Architectural Style:

- Processes divided into two (possibly overlapping) groups:
  - Server: process implementing a specific service (file system service, database service, ...)

- Client: process that requests a service from a server by sending it a request and subsequently waiting for the server's reply

– Request-reply protocol

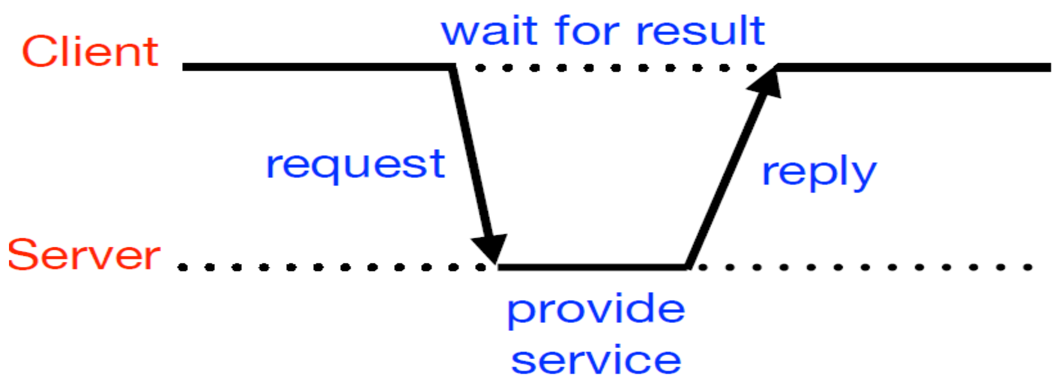


Figure 4 Client Server Interaction

Some Definitions:

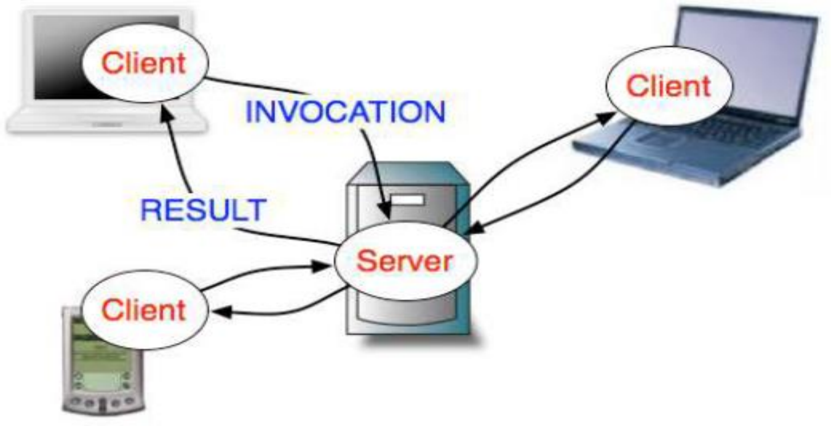
– "A *server* is a program (or collection of cooperating programs) that provides services and/or manages resources on the behalf of other programs (its *clients*)."

– "The client-server architecture is also termed as a network-computing structure because every request and their associated services are distributed over a network. In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested, process it and deliver the data packets requested back to the client. One special feature is that the server computer has the potential to manage numerous clients at the same time. Also, a single client can connect to numerous servers at a single timestamp, where each server provides a different set of services to that specific client. "

– "The client-server architecture is the most commonly used approach for data transfer. It designates a computer as a server and another one as a client. In a client-server architecture, the server needs to be online all the time, and get good connectivity. The server provides its clients with data, and can also receive data from clients. Some examples of widely used client-server applications are HTTP, FTP. All of these applications have a specific server-side functionality that implements the protocol"

Client - Server Interaction:

- Requests are sent in messages from clients to a server
    - o When a client sends a request for an operation to be carried out, we say that the client invokes an operation upon the server
- Replies are sent in messages from the server to the clients
- Remote invocation: a complete interaction between a client and a server (from the point when the client sends its request to when it receives the server's response)



Example: The Web as Client-Server Resource Sharing System

- The World Wide Web is an evolving and open system for publishing and accessing resources and services across the Internet
- For instance, through Web browsers (clients) users can
    - o retrieve and view documents of many types
    - o Listen to audio streams
    - o view video streams
    - o and in general, interact with an unlimited set of services

Web: Main Technological Components

1. The HyperText Markup Language (HTML) is a language for specifying the contents and layout of pages as they are displayed by Web browsers
2. Uniform Resource Locators (URLs) which identify documents and other resources stored as part of the Web
3. A client-server system architecture, with standard rules for interaction (the HyperText Transfer Protocol - HTTP) by which browsers and other clients fetch documents and other resources from Web servers
    - Web Browser and Web Server Example

On the Client and Server Role...

- A process can be both a client and a server, since servers sometimes invoke operations on other servers
- The terms "client" and "server" apply only to the roles played in a single request
- But in general, they are distinct concepts:
    - clients are active and server are passive (reactive)
    - server run continuously, whereas clients last only as long as the applications of which they form a part

On the Client-Server Role: Examples

- Example 1: A Web server is often a client of a local file server that manages the files in which the web pages are stored
- Example 2: Web servers and most Internet services are clients of the DNS service (which translates Internet Domain names to network addresses)
- Example 3: search engine
    - Server: it responds to queries from browser clients
    - Client: it runs (in the background) programs called web crawlers that act as clients of other web servers

Architectural Style: Peer-to-Peer (P2P)

- All the processes involved in a task or activity play similar roles, interacting cooperatively as peers without any distinction between client and server processes or the computers that they run on
- In practical terms, all peers run the same program and offer the same set of interfaces to each other

The aim of the P2P architecture is to exploit the resources (both data and hardware) in a large number of participating computers for the fulfilment of a given task or activity

Distributed Application Based on a P2P Architecture

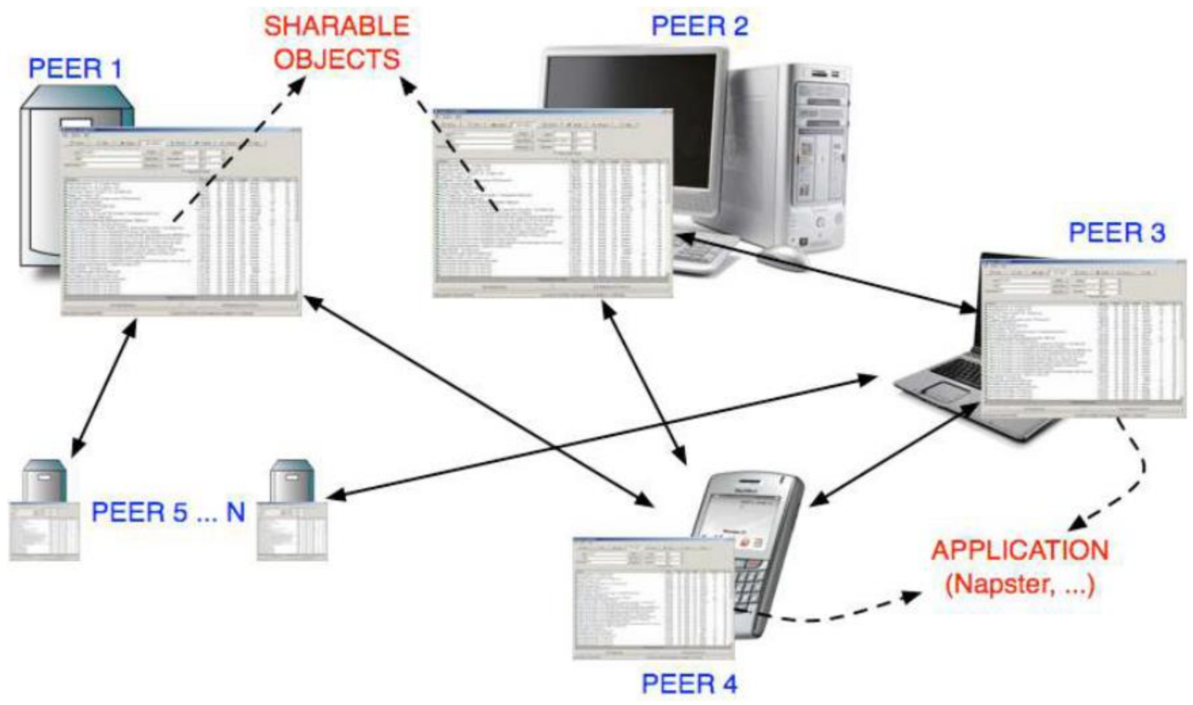


Figure 5 Distributed Application Based on a P2P Architecture

Architectural Models: Placement

- How are entities mapped on to the physical distributed infrastructure (i.e., what is their placement)?
- Physical distributed infrastructure usually consists of a potentially large number of machines interconnected by a network of arbitrary complexity
- Placement is crucial in terms of determining the properties of the distributed system, such as performance, reliability, and security
- Placement need to consider several aspects (machines, reliability, communication, ...) and there are few universal guidelines to obtaining an optimal solution!

Mapping of Services To Multiple Servers

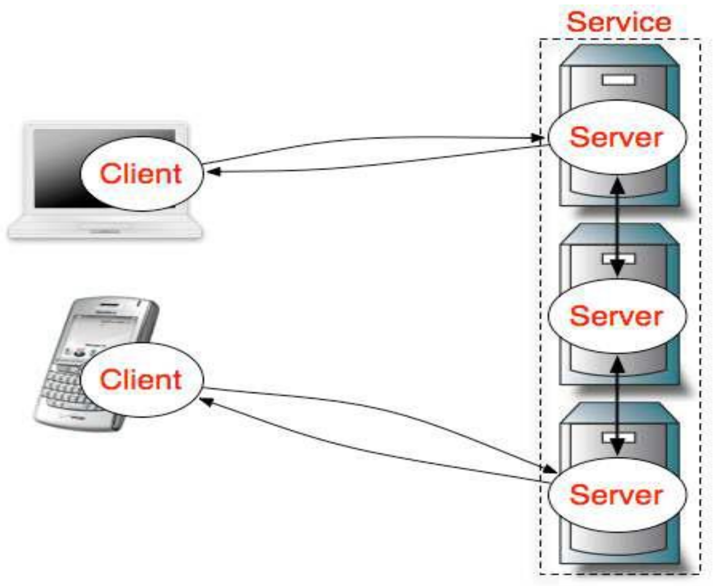


Figure 6 Mapping of services to multiple servers

Proxy Server and Caches

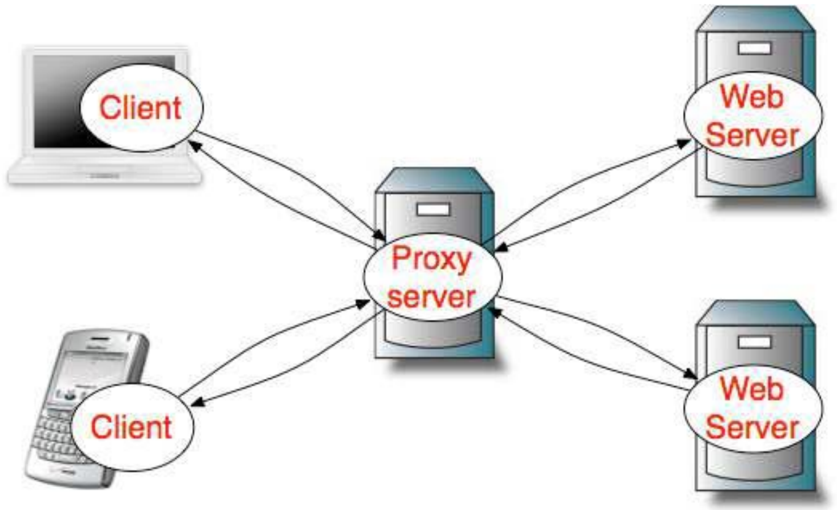


Figure 7 Proxy server and caches

Mobile Code

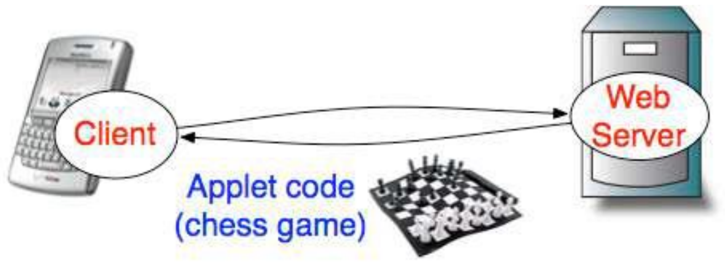


Figure 8 Mobile Code

Placement Strategy: Service Provided by Multiple Servers

- Services may be implemented as several server processes in separate host computers interacting as necessary to provide a service to client processes
- The servers may:
  - partition the set of objects on which the service is based and distributed them between themselves (e.g., Web servers)
  - they may maintain replicated copies of them on several hosts (e.g., SUN Network Information Service (NIS)).

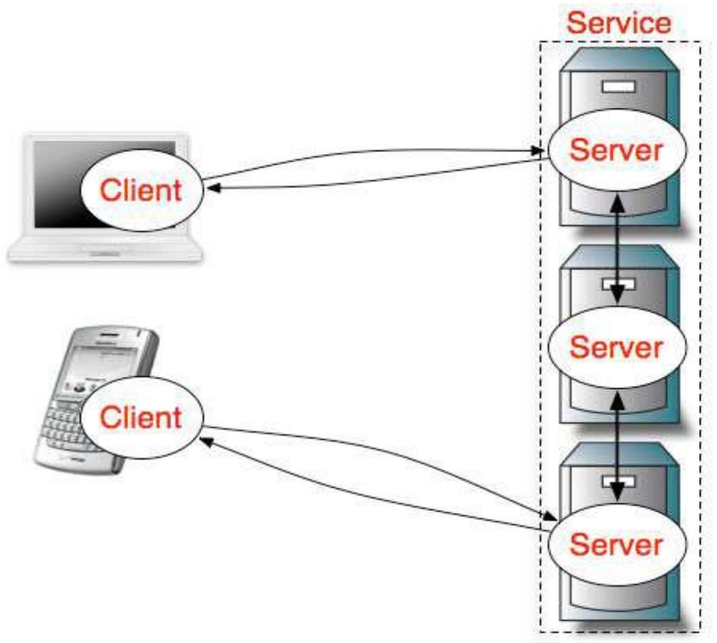


Figure 9 Service Provided by Multiple Servers

Placement Strategy: Proxy Servers and Caches

- A cache is a store of recently used data objects that is closer to one client or a particular set of clients than the objects themselves
- Example 1: Web browsers maintain a cache of recently visited pages and other web resources in the client's local file system
- Example 2: Web proxy server
    - To keep machines behind it anonymous (mainly for security)
    - To speed up access to a resource (via caching)

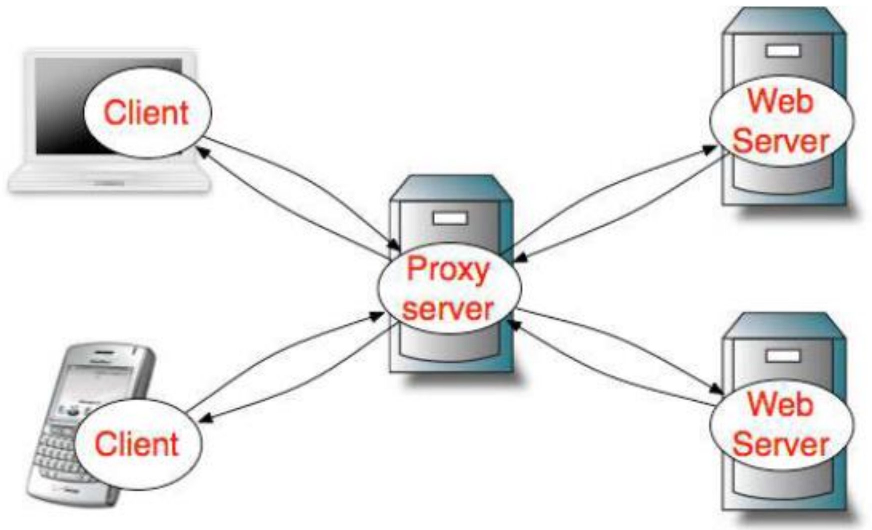
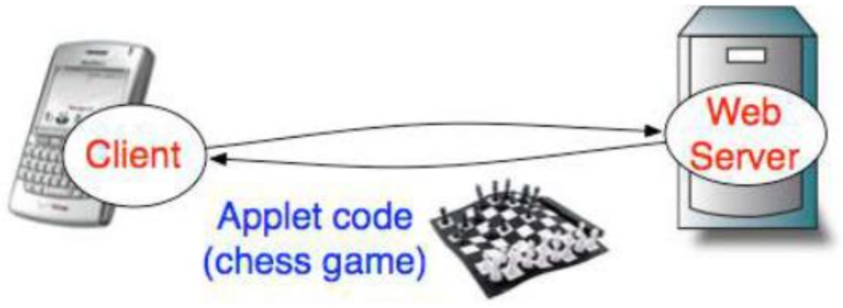


Figure 10 Proxy Servers and Caches

- provides a shared cache of web resources for the clients

Placement Strategy: Mobile Code

A) Client request results in the downloading of applet code



B) Client interacts with the applet

Client
Web Server

– An advantage of running the downloaded code locally is that it can give good interactive response since it does not suffer from the delays or variability of bandwidth associated with network communication

What's the Difference Between Peer to Peer and Client Server?

(These points are drawn from: https://www.resilio.com/blog/whats-the-difference-between-peer-to-peer-and-client-server)

"

Availability:

The most obvious problem that all client-server applications face is that server always has to be online and available. In case of any software, network or hardware problem, the service to all clients is affected. Therefore, you need to plan a server high availability solution in advance. High availability ensures that the system switches to a backup hardware or network if it's disrupted for any reason, and the service can continue to operate smoothly. This problem is quite complex since you need to keep data synchronized between your live machine and backup machine, and properly plan software and hardware updates in advance to support uninterrupted service operation.

High Load:

Another recurring problem with client-server applications is high load. A single powerful client that consumes data faster than the others could consume all the networking, disk operation and server CPU. You want all clients to have access to the server. Therefore, you need to limit clients to a certain consumption level, so each of them can get minimal server resources. This approach makes sure that the powerful client won't disrupt the other clients. But in reality, it usually means that the server always serves a client with the limitation in place, even if it's not overloaded and can operate faster.

Scalability:

Each server needs to be planned for the specific number of clients it will support. When the number of clients grows, the server CPU, memory, networking, and disk performance need to grow as well, and can eventually reach a point when the server stops operation. If you have more clients than a single server can serve, you probably need to deploy several servers. This means designing a system to balance and distribute load between servers, in addition to the high availability system we discussed previously.

And How Does Peer-to-Peer Stack Up?

Availability:

In a peer-to-peer world, each client is also the server. If the central machine is not available the service can be provided by any available client or a group of clients. The peer-to-peer system will find the group of best clients and will request service from them. This gives you service availability that doesn't depend on one machine and doesn't require the development of any high availability solution.

High Load:

The peer-to-peer, in comparison to client-server architecture, converts each node to a server that can provide service. Therefore, if a powerful client needs a lot of data, several other devices can provide it. Therefore, each client can download data at the fastest possible speed without any limitations.

Scalability:

Obviously, the more devices you have in a network the more devices will be able to participate in data delivery. They will participate in terms of network, CPU distributing this load from a central server. The more devices you have the less load will be on the central server.

Real Case Scenarios

Build distribution: Development companies struggle to deliver builds faster to remote offices across the globe, or within one office to hundreds of fast QA machines. In a client-server world, all remote offices will download build from a build machine. The speed limitation will be determined by the network channel that serves the build to all remote offices. A peer-to-peer approach would split the build into independent blocks that could travel between offices independently. This approach removes network limitation from the main office and combines the speed of all remote offices to deliver builds faster. Usually, you can get 3-5 times faster on a peer-to-peer architecture than on client-server.

Another issue is distributing build within a single office from a central server. The fast QA machines completely overload the central server network and CPU, bringing the central server to an unusable state. It is an almost unsolvable scalability issue. As we discussed above, a peer-to-peer approach is better when many clients need access to the data. Each QA machine can seed the data to other machines, keeping the server in a healthy state and delivering builds blazingly fast.

Data Delivery to Remote Offices

Delivering data to a remote office usually faces the issue of overloading the central server. Even if the speed of each office is not that fast, when you have many of them it adds up and requires huge bandwidth channels to the central office. A problem can occur when you need to deliver software or OS updates as well as other data such as documents, video, or images. The peer-to-peer approach solves that by allowing each remote office to participate in data delivery. This reduces the load on the central server, and significantly reduces central server and networking requirement.

Another long-lasting problem peer to peer solves is when the data needs to be distributed to several machines within the office. The previous approach was to either download from the central server which would increase load even more, or develop a two-step distribution policy: deliver data to the central server within the remote location, and then copy data locally. The peer-to-peer solves that naturally by finding the best source for the data, whether it is a local server or remote server. Once the block of the data is present in the remote office, it won't necessarily be downloaded from the central data center."

Next : We will look at a Design problem of a Client-Server System for Banking

- 2 - Tier Vs 3 -Tier
- Thin VS Thick Client Model
    - Two-tier client-server architecture with thin clients
    - Two-tier client-server architecture with fat clients
    - Multi-tier client-server architecture