Distributed Systems Notes

Section 1: Introduction

What is a Distributed System:

- A collection of independent computers that appears to its users as a single coherent system.
    - Number of components
    - Communication between components (message passing)
    - Achieve more than each machine can individually

A computer network is just a collection of computers passing messages. E.g the internet is a computer network. It does not appear as a single system to users.

Advantages of a distributed System:

- Resource sharing (disks, printers, files, processing power)
- Availability, scalability, reliability.

Disadvantages of a distributed System:

- Concurrency
- Heterogeneity
- No global clock
- Independent failures

The internet:

The internet is a large number of interconnected computer networks. Includes features such as protocol for message passing and has services like world wide web(www) and email and file transfer.

Intranets:

Portion of internet that is separately administered by organizations. Connected to internet via a router. Firewalls to protect intranet from unauthorized messages.

Distributed System Challenges:

Heterogeneity: Distributed systems use hardware and software resources of varying characteristics. E.g OS is linux or windows, or programming language. Can solve problem by using middleware or agreeing on standard data/protocols.

Middleware: software layer between the distributed application and operating systems. E.g Distributed file system, RPC (procedural language), RMI. (Object-oriented language).

Openness: Ability to extend the system in different ways be adding hardware or software resources. E.g adding interfaces.

Security:

- Confidentiality: Protect against unauthorized users
- Integrity: Protect against alteration and corruption
- Availability: protection against interference with access

Security mechanisms include encryption, authentication (passwords, keys), authorization (access control)

Scalability: System needs to handle the growth of users and data.

- Cost of physical resources
- Controlling performance loss, avoiding bottle necks (good algorithms)
- Available resources

Failure Handling:

- Detecting, Masking (message retransmission), Tolerating (report failure to user), recovery (fix corrupted data).

Concurrency: Multiple clients accessing the same resource at same time. (Use semaphores)

Transparency: Hiding the components of a distributed system for the user and application programmer. (Access, location, concurrency, failure and scaling transparency)

Section 2: Models

Communication Paradigms (Low level to high level):

Interprocess communication: Multicast (message to multiple users), socket con.

Remote Invocation: Call a remote operation between dist. entities. RMI, RPC.

Indirect communication: Space uncoupling (senders don't know who's sending). Time uncoupling (senders and receivers don't need to exist at same time).

Roles and Responsibilities:

Client: Initiates connection to other process.

Server: Receives connection, offers service.

Peer: Client or server, connect to and receive connection from other peers.

Placement:

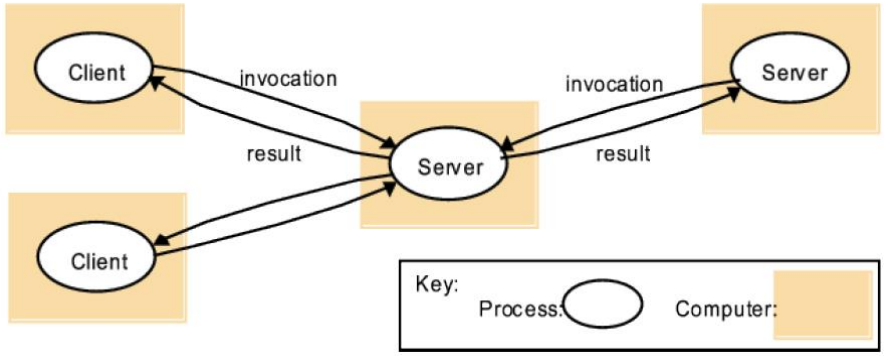
Caching: Storing data at places closer to client speeds up responses.

Mobile code: Transferring code to location most efficient. E.g running complex query on another machine. Forcing client to execute code etc.

Mobile agents: Code and data together executes on the client PC. Also, agent may check for updates to ensure software on client's PC is up to date.

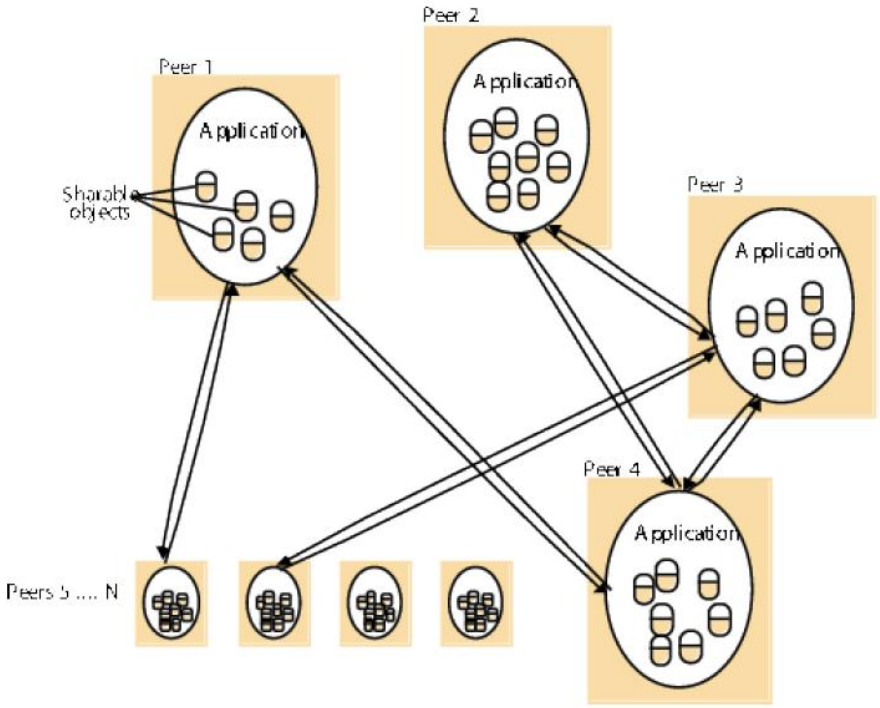
Architectural Patterns:

Client-Server: Client invokes services in server and results are returned.



Server and a bunch of clients who can make use of the server. ***Client's don't talk to one another****. Communication happens through the server.*

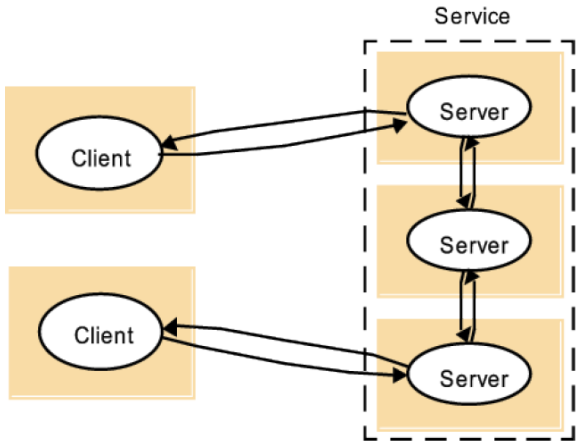
Peer-to-Peer: Each process in the system plays a similar role as client or server.



Every component can make a connection to every other component as well as receive a connection. ***There is no central server****. Can be* ***more secure than client-server*** *as it is* ***not susceptible to a single point of failure****. Bringing down all peers harder than a central server. Peer is not more important than another peer.*

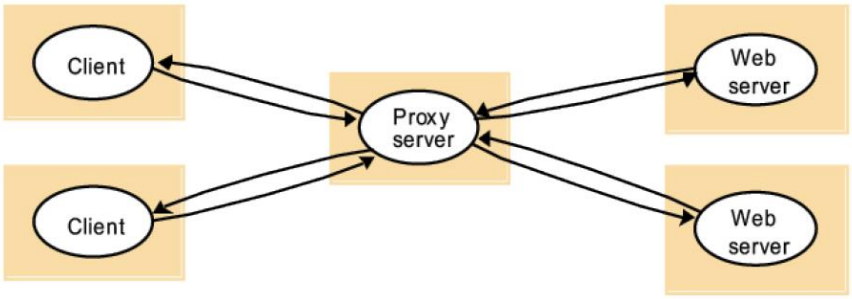
Distributed Architecture Variations:

Service Provided by multiple servers:



- *Objects may be replicated across servers*
- *Helps with load balancing*
- *Can offer service closer to client*
- *Survives failure of one server*

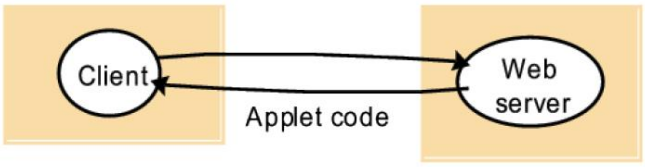
Proxy servers and caches:



- *Cache is a store of recently used objects closer to client*
- *Helps with load balance*
- *Speeds requests up*
- *Can use in authentication*

Mobile Code and Agents:

a) client request results in the downloading of applet code



b) client interacts with the applet

- *Downloaded and executed by the client.*
- *Less resources used by server -> scalable.*