# Characterization of Distributed Systems

## Introduction

A distributed system is the one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages.

## Characteristics of Distributed Systems

### Concurrency

In a network of computers, concurrent programs execution is the norm. Concurrency refers to the system’s ability to handle multiple access and use of shared resources. This is important because if there is no measure implemented, then data can get corrupted or lost by two nodes making different changes to the same resource causing an error in the program execution. One way to counteract these errors is to implement a locking mechanism making a node unable to access a resource whilst it is being used by another node.

### Openness

The openness of a distributed system is defined as the difficulty involved to extend or improve an existing system. This characteristic allows us to reuse a distributed system for multiple functions or to process varying sets of data.

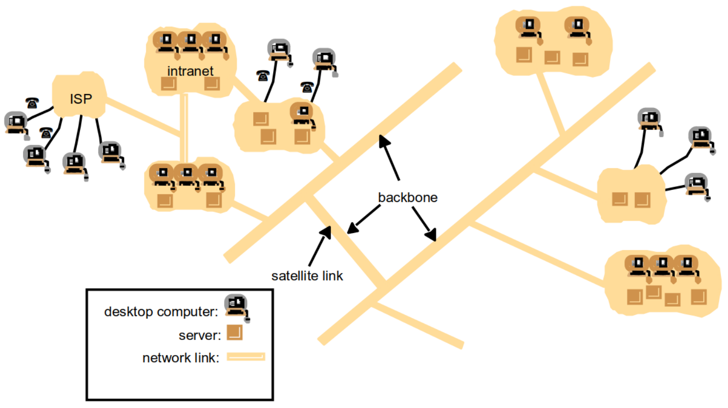
### Fault Tolerance

Due to a distributed system having many computers comprised of different aged hardware, it is very likely for a part to fail in such a way that a node can no longer operate. Fault Tolerance is the ability for the system to handle such failures, this is achieved by using recovery and redundancy. Recovery is where a component will act in a predictable, controlled way if it relies on a component. Redundancy is where crucial systems and processes will have a backup that takes over if a system fails.

## Q1 Examples of Distributed Systems

## Internet

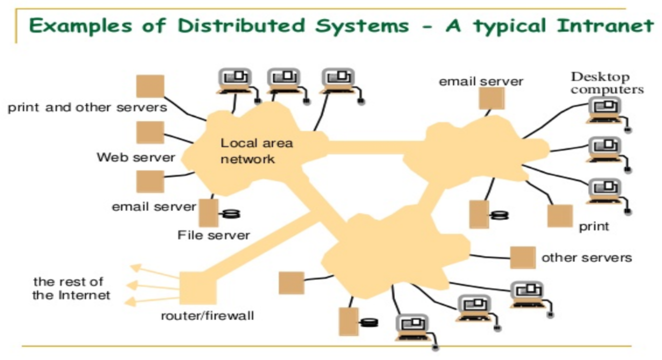
Figure . A typical portion of the Internet



The Internet is a very large distributed system. It enables users, wherever they are, to make use of services like www, email, file transfer, etc. The set of services is open-ended i.e. it can be extended by the addition of server computers and new types of services. The modern Internet is a vast interconnected collection of computer networks of many different types, with a range of types increasing all the time.

## Intranets

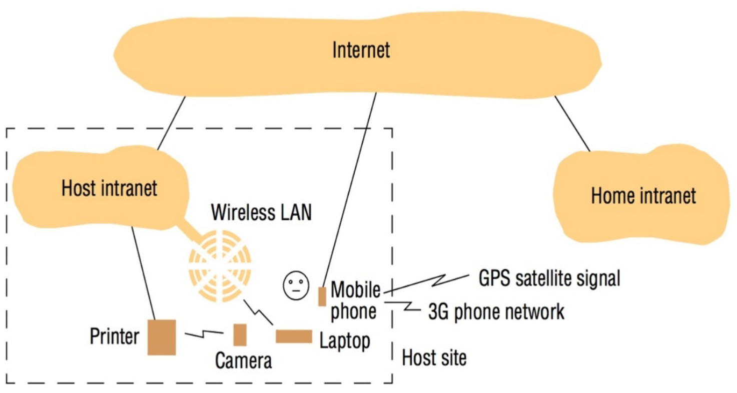
A typical Intranet



- An Intranet is a portion of the internet that is separately administered and has a boundary that can be configured to enforce local security policies.
- It may be composed of several LANs linked by backbone connections.
- The network configuration of a particular intranet is the responsibility of the organization that administers it.
- An intranet is connected to the Internet via a router, which allows the users to use the services available on the Internet.
- A firewall is used to protect the intranet by preventing unauthorized messages from leaving or entering.
- Some organizations do not wish to connect their internal networks to the Internet at all.
- E.g. police and other security and law enforcement agencies are likely to have at least some internal networks that are isolated from the outside world.
- These organizations can be connected to the Internet to avail themselves of the services by dispensing with the firewall.
- The main issues arising in the design of components for use in intranets are:
  - File services are needed to enable users to share data
  - Firewalls should ensure legitimate access to services
  - The cost of installation and support should be minimum

## Mobile and Ubiquitous Computing

Figure: Portable and handheld devices in a distributed system



- Integration of portable computing devices like Laptops, smartphones, handheld devices, pagers, digital cameras, smartwatches, devices embedded in appliances like refrigerators, washing machines, cars, etc. with the distributed systems became possible because of the technological advances in device miniaturization and wireless networking.
- These devices can be connected to each other conveniently in different places, making mobile computing possible.
- In mobile computing, users who are away from the home intranet, are still allowed to access resources via the devices they carry.
- Ubiquitous computing is the harnessing of many small, cheap computational devices that are present in users' physical environments, including home, office, and others.
- The term ubiquitous is intended to suggest that small computing devices will eventually become so pervasive in everyday objects that they are scarcely noticed.
- The presence of computers everywhere is useful only when they can communicate with one another.
- E.g. it would be convenient for users to control their washing machine or their entertainment system using a “Universal remote control” device at home.
- The mobile user can get benefit from computers that are everywhere.
- Ubiquitous computing could benefit users while they remain in a single environment such as the home, office, or hospital.
- The figure shows a user who is visiting a host organization. The user's home intranet and the host intranet at the site that the user is visiting. Both intranets are connected to the rest of the Internet.

# Q2 Challenges

## Heterogeneity

The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks.
Heterogeneity (that is, variety and difference) applies to all of the following:

- Networks: Local network, the Internet, wireless network, satellite links, etc.

- Hardware devices: Computers, tablets, smartphones, embedded devices, etc.
- Operating systems: Windows, Linux, Mac, Unix, etc.
- Programming languages: Java, C/C++, Python, PHP, etc.
- Different roles like software developers, designers, system managers, etc.

Different programming languages use different representations for characters and data structures such as arrays and records. These differences must be addressed if programs written in different languages are to be able to communicate with one another. Programs written by different developers cannot communicate with one another unless they use common standards, for example, for network communication and the representation of primitive data items and data structures in messages. For this to happen, standards need to be agreed upon and adopted – as have the Internet protocols.

## Openness

- The openness of a computer system is the characteristic that determines whether the system can be extended and re-implemented in various ways.
- The openness of distributed systems is determined primarily by the degree to which new resource-sharing devices can be added.
- Openness cannot be achieved unless the specification and documentation of the key software interfaces of the components of a system are made available to software developers. If the well-defined interfaces for a system are published, it is easier for developers to add new features or replace sub-systems in the future.

## Security

- Many of the information in distributed systems have a high intrinsic value to their users.
- Their security is therefore of considerable importance.
- Security for information resources has three components:
    - **Confidentiality:** protection against disclosure to unauthorized individuals
    - **Integrity:** protection against alteration or corruption of data
    - **Availability:** protection against interference with the means to access the resources

## Scalability

- A system is described as **scalable** if it will remain effective when there is a significant increase in the number of resources and the number of users.
- The number of computers and servers on the Internet has increased dramatically.
- The design of scalable distributed systems presents the following challenges:
  - Controlling the cost of physical resources. Ex: File server expanding.
  - Controlling the performance loss. Ex: DNS nametable algorithms.
  - Preventing software & hardware resources from running out. Ex: IP address, CPU & Memory burst.
  - Avoiding performance bottlenecks. Ex: caching and replication, unexpected program failure, no proper error handling mechanism, etc.

## Failure Handling

- When faults occur in hardware or software, programs may produce incorrect results or may stop before they have completed the intended computation.
- Failures in a distributed system are partial, i.e. some components fail while others continue to function.
- The following are techniques for dealing with failures:
  - **Detecting failures:** checksum and investigating servers under the crash.
  - **Masking failures:** Message re-transmission, writing data on multiple areas of the disk.
  - **Tolerating failures:** Web browser trying to re-connect to the API.
  - **Recovery from failure:** Rollback or backup state.
  - **Redundancy:** Routing (load-balancers), replication of database on multiple locations.

## Concurrency

- Both services and applications provide resources that can be shared by clients in a distributed system.

- There is therefore a possibility that several clients will attempt to access a shared resource at the same time.
- For example, a data structure that records bids for an auction may be accessed very frequently, multiple times when it gets close to the deadline time.
- The process that manages a shared resource could take one client request at a time. But that approach limits throughput.
- Therefore services and applications generally allow multiple client requests to be processed concurrently.

## Transparency

Transparency is defined as the concealment from the user and the application programmer of the separation of components in a distributed system so that the system is perceived as a whole rather than as a collection of independent components. In other words, distributed systems designers must hide the complexity of the systems as much as they can. Some terms of transparency in distributed systems are:

- **Access transparency:** Hide differences in data representation and how a resource is accessed.
- **Location transparency:** Enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address).
- **Concurrency transparency:** Hide that a resource may be shared and accessed by multiple users at the same time.
- **Replication transparency:** Hide that resources may be copied in several places.
- **Failure transparency:** Enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.
- **Mobility transparency** allows the movement of resources and clients within a system without affecting the operation of users or programs.
- **Performance transparency** allows the system to be reconfigured to improve performance as loads vary.
- **Scaling transparency** allows the system and applications to expand in scale without change to the system structure or the application algorithms.

**3. Define Architecture Model. Mention its goal & explain the following with example:**
**a) Mobile Code b) Mobile Agent**
**c) Proxy Server & Cace**

# System Models

## Types of System Models

### Architectural Models

It describes a system in terms of the computational and communication tasks performed by its computational elements; the computational elements being individual computers or aggregates of them supported by appropriate network interconnections.

### Fundamental Models

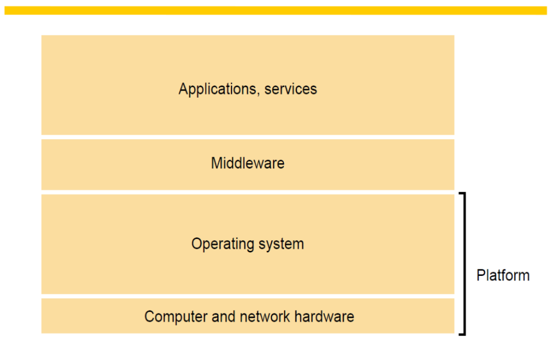
It takes an abstract perspective, in order to examine individual aspects of a distributed system. It is a model that contains only the essential ingredients that we need to consider in order to understand and reason about some aspects of a system's behavior.

**Three important aspects of Fundamental Models:**

- **Interaction Models:** It considers the structure and sequencing of the communication between the elements of the system.
- **Failure Models:** It considers the ways in which a system may fail to operate correctly.
- **Security Models:** It considers how the system is protected against attempts to interfere with its correct operation or to steal its data.

## Q3. Types of hardware and software resources that can be shared in a distributed system

Figure 2.7
Software and hardware service layers in distributed systems



## Hardware

**CPU:** Most servers, such as filer servers, do some computation for their clients, hence their CPU is a shared resource.

**Memory:** Cache server (holds recently-accessed web pages in its RAM, for faster access by other local computers).

**Disk:** File server, virtual disk server, video on demand server, etc

**Printer:** Networked printers accept print jobs from many computers. managing them with a queuing system.

**Network Capacity:** Packet transmission enables many simultaneous communication channels (streams of data) to be transmitted on the same circuits.

**Graphics card:** As machine learning models require heavy computation hardware and graphics cards are a costly piece of hardware, making it a shared resource will make it cost-effective.

## Software

**Web Page:** Web servers enable multiple clients to share read-only page content (usually stored in a file, but sometimes generated on the fly).

**File:** File servers enable multiple clients to have read-write access to the same file.

**Object:** Possibilities for software objects are limitless. E.g. shared whiteboard, room booking system, etc.

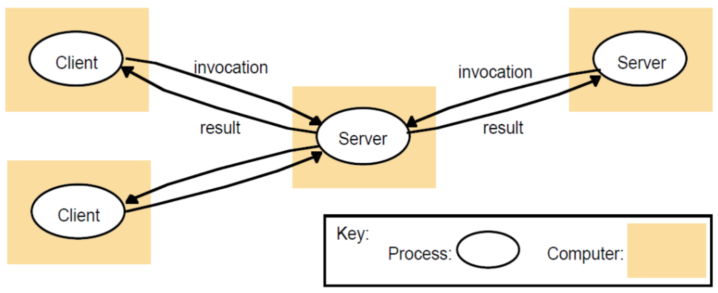
**Database:** Databases are used to store useful data about and for the users. Databases can be accessed and used concurrently in a distributed system.

**Video / Audio Stream:** Servers can store entire videos on disk and deliver them at playback speed to multiple clients simultaneously.

**Exclusive Lock:** A system-level object provided by a lock server, enabling several clients to coordinate their use of a resource (such as a printer that does not include a queuing scheme).

## Client-Server Model

Figure 2.3
Clients invoke individual servers



The system is structured as a set of processes, called servers, that offer services to the users, called clients.

- The client-server model is usually based on a simple request/reply protocol, implemented with send/receive primitives or using remote procedure calls (RPC) or remote method invocation (RMI)
- The client sends a request (invocation) message to the server asking for some service
- The server does the work and returns a result (e.g. the data requested) or an error code if the work could not be performed.
- A server can itself request services from other servers; thus, in this new relation, the server itself acts as a client.
- For example: Search engines, which enable users to look up summaries of information available on web pages at sites throughout the Internet.
- These summaries are made by programs called web crawlers, which run in the background at a search engine site using HTTP requests to access web servers throughout the Internet.
- Thus a search engine is both a server and a client: it responds to queries from the browser clients and it runs web crawlers that act as clients of other web servers.

## Q4. Discuss the Software Layers of distributed system architectural model.

- Software architecture refers to services offered and requested between processes located in the same or different computers.

- It is the structure of software as layers or modules in a single computer.
    - **Layer:** A group of related functional components.
    - **Service:** Functionality provided to the next layer.
- **Platform:** The lowest-level hardware and software layers are often referred to as a platform for distributed systems and applications.
    - These low-level layers provide services to the layers above them, which are implemented independently in each computer.
    - These low-level layers bring the system's programming interface up to a level that facilitates communication and coordination between processes.
    - Ex: Intel x86/Windows, Intel x86/Linux, etc.
- **Middleware:** It is a layer of software whose purpose is:
    - To provide a convenient programming model to application developers.
    - To mask heterogeneity presented in distributed systems and provide interoperability between the lower layer and upper layer.

## Q5. What are variations of the client-server model?

### There are four main categories of client-server computing:

- **One-Tier architecture**: It consists of a simple program running on a single computer without requiring access to the network. User requests don't manage any network protocols, therefore the code is simple and the network is relieved of the extra traffic.
- **Two-Tier architecture**: It consists of the client, the server, and the protocol that links the two tiers. The GUI code resides on the client host and the business logic resides on the server host.
- **Three-Tier architecture**: It consists of a presentation tier, which is the User Interface layer, the application tier, which is the service layer that performs detailed processing, and the data tier, which consists of a database server that stores information.
- **N-Tier architecture:** It divides an application into logical layers, which separate responsibilities and manage dependencies, and physical tiers, which run on separate machines, improve scalability and add latency from the additional network communication.

## Q6. Describe the Interaction Model of a distributed system

Interaction models are for handling time i. e. for process execution, message delivery, clock drifts, etc.

### There are two variants of the Interaction Model

**Synchronous Distributed Systems:**

- The time taken to execute each step of a process has known lower and upper bounds.
- Each message transmitted over a channel is received within a known bounded time.
- Each process has a local clock whose drift rate from real-time has a known bound.

**Asynchronous Distributed Systems:**

- The time taken to execute each step of a process can be arbitrarily long.
- Each message transmitted over a channel may be received after an arbitrarily long time.
- Each process has a local clock whose drift rate from real-time may be arbitrarily long.

## Q7. Write the design requirements for Distributed architectures.

### i) Performance Issues

Performance issues arising from the limited processing and communication capacities of computers and networks are considered under the following subheading:

- **Responsiveness:** Time the system takes to process a request. How quickly the system acknowledges a request.
- **Throughput:** It is the rate at which computational work gets done.
- **Balancing of Computational Loads:** Using several computers to host a single service. Ex: Using applets on clients, to remove the load on the server.

### ii) Quality of service

The ability of systems to meet deadlines. It depends on the availability of the necessary computing and network resources at the appropriate time.
The main properties of the quality of the service are:

- Reliability
- Security
- Adaptability
- Performance

### iii) Use of caching and replication

Distributed systems overcome the performance issues by the use of data replication and caching. Cached copies of resources should be kept up-to-date when the resource at a server is updated.

### iv) Dependability Issues

Dependable applications should continue to function correctly in the presence of faults in hardware, software, and networks.

### v) Fault Tolerance

Dependable applications should continue to function in the presence of faults in hardware, software, and networks.

## Q8. Describe the failure model of the distributed system

- Failures can occur both in processes and communication channels. The reason can be both software and hardware faults.
- Fault models are needed in order to build systems with predictable behavior in case of faults (systems that are fault-tolerant).
- Such a system will function according to the predictions, only as long as the real faults behave as defined by the “fault model”.

### Variants of the Failure Model

**Omission Failures:**

- A process or channel fails to perform the actions that it is supposed to do.

- Process Omission Failure:
    - A process completes a send-operation, but the message is not put into the outgoing message buffer.
    - A message is put into a process's incoming message buffer, but the process does not receive it.
- Communication Omission Failure:
    - It is a 'fail-stop' failure, which means the server only exhibits crash failures, but at the same time, any correct server in the system can detect that this particular server has failed.

**Arbitrary Failures:**

- The term arbitrary or Byzantine failure is used to describe the worst possible failure semantics, in which any type of error may occur.
- An arbitrary failure of a process is one in which it arbitrarily omits intended processing steps or takes unintended processing steps.
- Process or channel may exhibit arbitrary behaviour when failing:
    - To send/receive arbitrary messages at arbitrary intervals.
    - A process may halt or perform "faulty" steps.

**Timing Failures:**

- Timing failures are applicable in synchronous distributed systems where time limits are set on process execution time, message delivery time, and clock drift rate.
- Timing Failures can be classified as:

| Class of Failure | Affects | Description |
|---|---|---|
| Clock | Process | Process’s local clock exceeds the bounds on its rate of drift from real-time. |
| Performance | Process | Process exceeds the bounds on the interval between two processing steps. |
| Performance | Channel | A message’s transmission takes longer than the stated bounds. |

**Masking Failures:**

- Knowledge of the failure characteristics of a component can enable a new service to be designed to mask the failure of the components on which it depends.
- A service masks a failure either by hiding it altogether or by converting it into a more acceptable type of failure.
- Example: Multiple servers that hold replicas of data can continue to provide a service when one of them crashes.