CHARACTERIZATION OF DISTRIBUTED SYSTEMS

From Chapter 1 of Distributed Systems Concepts and Design,4th Edition,

By G. Coulouris, J. Dollimore and T. Kindberg

Published by Addison Wesley/Pearson Education June 2005

Topics

- Defining Distributed Systems
- Resource sharing and the Web
- Design Challenges of Distributed Systems

Defining Distributed Systems

- Various definition of distributed systems have been given in the literature, for example:
  - A collection of logically related data that is distributed over different processing nodes of computer network.
- Definition above does not provide all characteristics of distributed systems.

Defining Distributed Systems

- It is difficult to find a definition that provides all characteristics of distributed systems.
- Another way is a definition that gives a loose characterization for distributed systems such as:
  - A distributed system is a collection of independent computers that appear to the users of the system as a single computer.
- With any definition, sharing of resources is a main motivation for constructing distributed systems.

Defining Distributed Systems

- In this course, we define distributed systems more precisely as :
  - A distributed system is one in which hardware or software components located at networked computers communicate and coordinate their actions only by message passing.
- Definition above covers the entire range of distributed systems in which networked computers can usefully be deployed.

Defining Distributed Systems

- Networks of computers are everywhere!
- Examples of networks of computers are:
  - Mobile phone networks
  - Corporate networks
  - Factory networks
  - Campus networks
  - Home networks
  - In-car networks
  - On board networks in aero planes and trains

Defining Distributed Systems

- Our definition of distributed systems has the following significant consequences:
  - Concurrency
    - Tasks carry out independently
  - No global clock
    - Tasks coordinate their actions by exchanging messages

Defining Distributed Systems

- Independent Failures
  - Faults in the network result in the isolation of the computers that are connected to it.
  - Each component of the system can fail independently, leaving the others still running.

General Examples of Distributed Systems

- Internet
- Intranets
- Mobile networks

General Examples of Distributed Systems

- The Internet
  - The Internet is a vast interconnected collection of computer networks of many different types.
  - Multimedia services are available in the Internet enabling users to access audio and video data including music, radio, TV channels, phone, and video conferencing.

(Figure 1)

General Examples of Distributed Systems

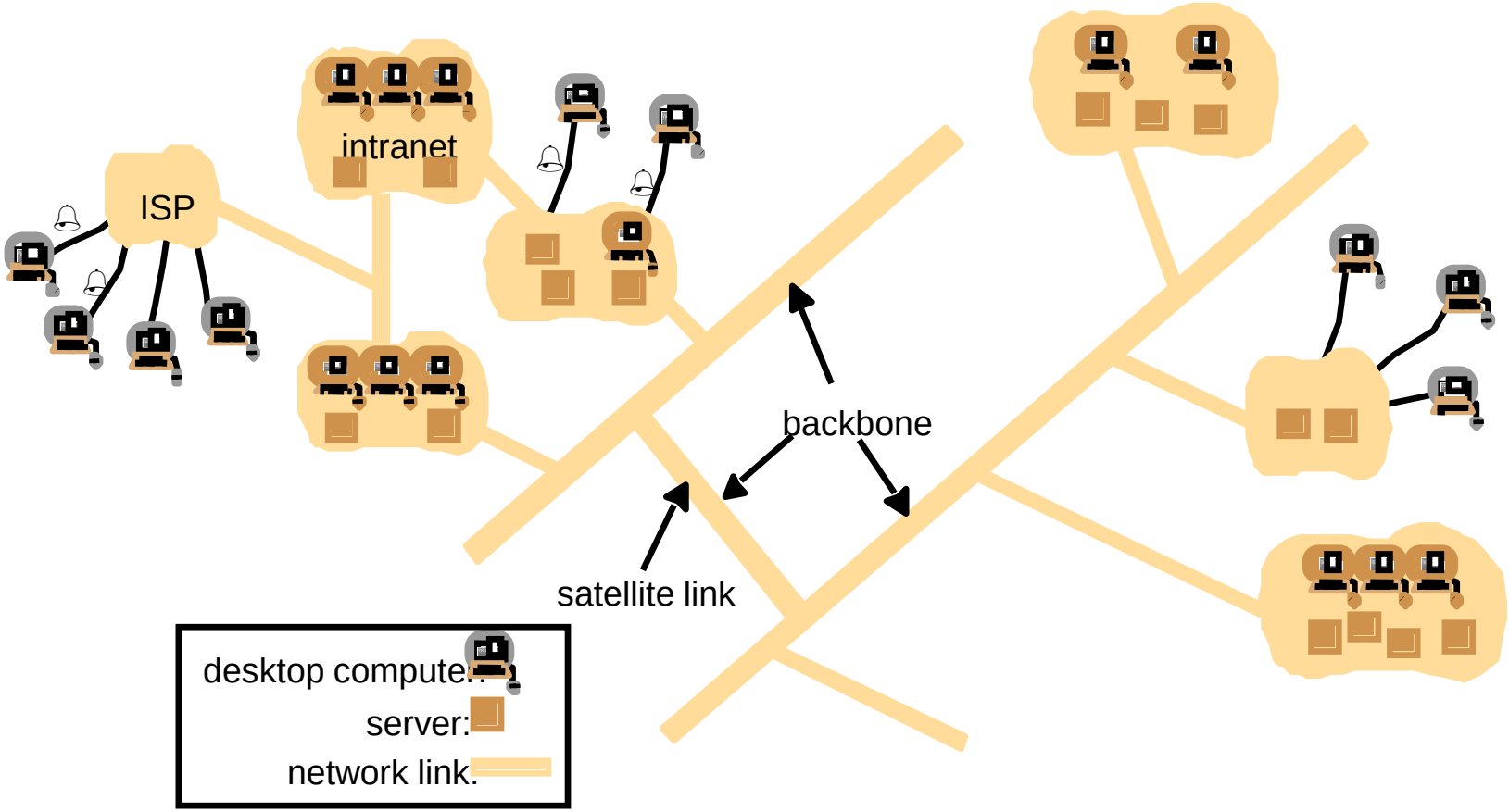

Figure 1. A typical portion of the Internet

Couloris,Dollimore and Kindberg Distributed Systems: Concepts & Design Edn. 4 , Pearson Education 2005

General Examples of Distributed Systems

- 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.

(Figure 2)

General Examples of Distributed Systems

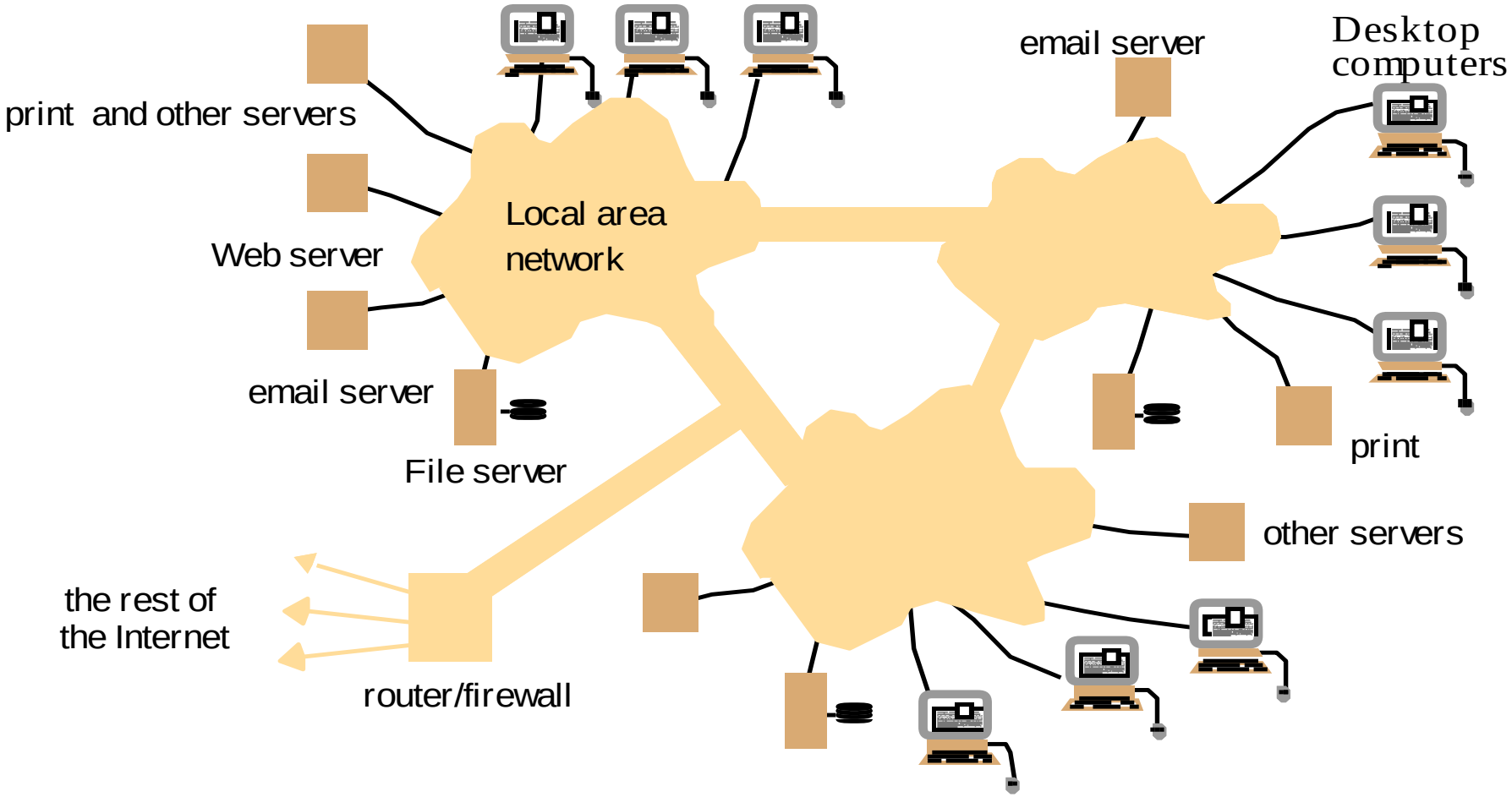

Figure 2. A typical Interanet

General Examples of Distributed Systems

- Mobile networks
  - Technological advances in device miniatorization and wireless networking have led increasingly to the integration of small and portable computing devices into distributed systems.
  - These devices include:
    - Laptop computers

General Examples of Distributed Systems

- Handheld devices
  - Personal digital assistants(PDAs)
  - Mobile phones
  - Pagers
  - Video cameras
  - Digital cameras

General Examples of Distributed Systems

- Wearable devices
  - Smart watches with functionality similar to a PDA
- Devices embedded in appliances
  - Washing machines
  - Hi-fi systems
  - Cars
  - Refrigerators

(Figure 3)

General Examples of Distributed Systems

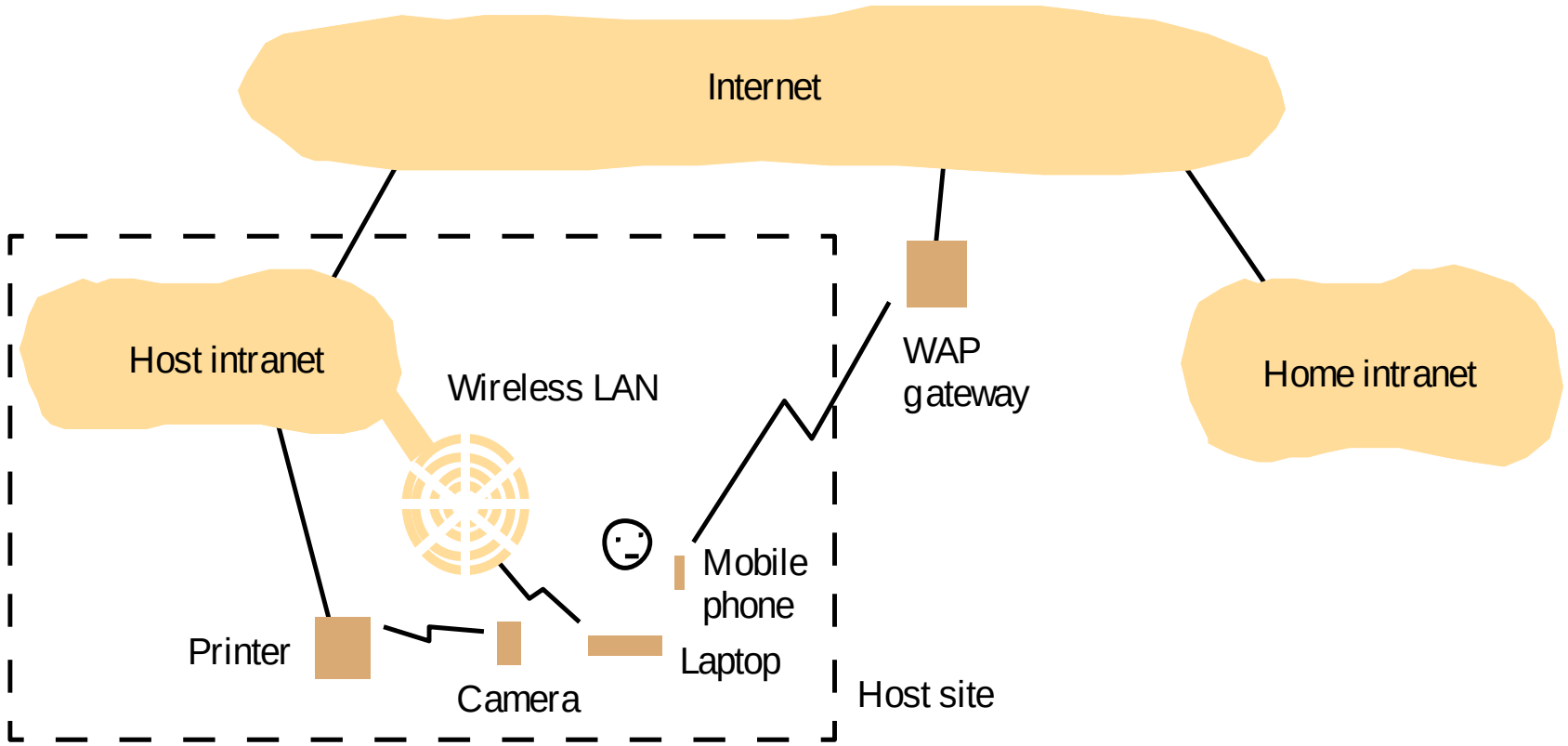

Figure 3. Portable and handheld devices in a distributed system.

Couloris,Dollimore and Kindberg Distributed Systems: Concepts & Design Edn. 4 , Pearson Education 2005

Other Examples of Distributed Systems

- Cluster
  - A type of parallel or distributed processing system, which consists of a collection of interconnected stand-alone computers cooperatively working together as a single, integrated computing resource. The computers may be standard per uniprocessor or multiprocessor.
  - A cluster can be used for providing highly scalable services such as search engines provide for users all over the Internet.

Other Examples of Distributed Systems

- Grid
  - A type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements.

Resource sharing and the Web

- The World Wide Web
  - The World Wide Web is an evolving system for publishing and accessing resources and services across the Internet.

(Figure 4)

Resource sharing and the Web

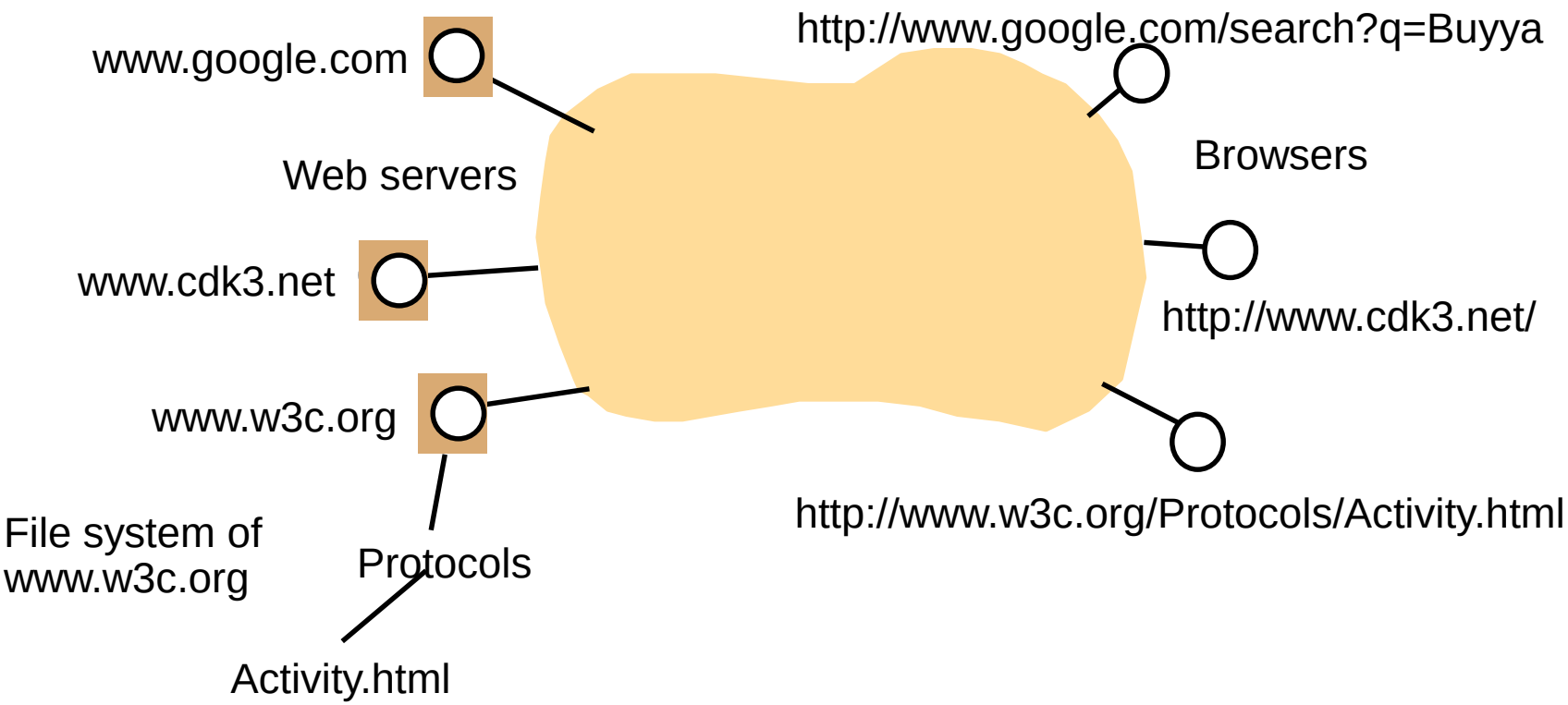

Figure 4. Web servers and web browsers.

Design Challenges of Distributed Systems

- Designers of distributed systems need to take the following challenges into account:
  - Heterogeneity
    - Heterogeneous components must be able to interoperate.
  - Openness
    - Interfaces should allow components to be added or replaced.
  - Security
    - The system should only be used in the way intended.

Design Challenges of Distributed Systems

- Scalability
  - System should work efficiently with an increasing number of users.
  - System performance should increase with inclusion of additional resources.
- Failure handling
  - Failure of a component (partial failure) should not result in failure of the whole system.

Design Challenges of Distributed Systems

- Transparency
  - Distribution should be hidden from the user as much as possible.

Heterogeneity

- Heterogeneous components that must be able to interoperate, apply to all of the following:
  - Networks
  - Hardware architectures
  - Operating systems
  - Programming languages

Heterogeneity

- Examples that mask differences in network, operating systems, hardware and software to provide heterogeneity are
  - Middleware
  - Internet protocols
  - Mobile code

Heterogeneity

- Middleware
  - Middleware applies to a software layer.
  - Middleware provides a programming abstraction.
  - Middleware masks the heterogeneity of the underlying networks, hardware, operating systems and programming languages.
  - The Common Object Request Broker (CORBA) is a middleware example.

Heterogeneity

- Mobile code
  - Mobile code is the code that can be sent from one computer to another and run at the destination.
  - Java applets are the example of mobile codes.

Heterogeneity

- Virtual machine
  - Virtual machine provides a way of making code executable on any hardware.

Openness

- Distributed systems must be extensible.

- Openness of a computer system is the characteristic that determines whether the system can be extended and re-implemented in various ways.

Openness

- The first step in openness is publishing the documentation of software components and interfaces of the components to make them available to software developers.

Security

- Security of a computer system is the characteristic that the resources are accessible to authorized users and used in the way they are intended.

- Security for information resources has three components:
  - Confidentiality
    - Protection against disclosure to unauthorized individual.

Security

- Integrity
  - Protection against alteration or corruption.
- Availability
  - Protection against interference with the means to access the resources.

Security

- Security Mechanisms are:
  - Encryption
  - Authentication
  - Authorization

Security challenges

- Denial of service attacks
  - Denial of service attacks is an attempt to make a computer resource unavailable to its intended users.
- Security of mobile code
  - Mobile code needs to be handled with care.
    - E.g. receiving an executable program as an electronic mail attachment to display an interesting picture but in reality it may access local resources, or perhaps be part of a denial of service attack.

Scalability

- Scalable distributed systems operate effectively and efficiently at many different scales, ranging from a small Intranet to the Internet.

- Scalable distributed systems remain effective when there is a significant increase in the number of resources and the number of users.

Scalability

- Challenges of designing scalable distributed systems are:
  - Controlling the cost of physical resources
    - Cost should linearly increase with the system size.
  - Controlling the performance loss
    - For example, in hierarchically structured data, search performance loss due to data growth should not be beyond O(*log n*), where n is the size of data.

Scalability

- Preventing software resources running out
  - An example is the numbers used as Internet addresses (IP)(32 bit->128-bit)
  - Y2K like problem.
- Avoiding performance bottlenecks
  - Using decentralized algorithms to avoid having performance bottlenecks.

- Caching and replication in Web are examples of providing scalability.

Failure handling

- Failures in distributed systems are partial, that is some components fail while others continue to function.

- Techniques for dealing with failures:
  - Detecting failures
    - E.g. Checksums
  - Masking failures
    - E.g. Retransmission of corrupt messages
    - E.g. File redundancy

Failure handling

- Tolerating failures
  - E.g. Exception handling
  - E.g. Timeouts
- Recovery from Failure
  - E.g. Rollback mechanisms
- Redundancy
  - E.g. Redundant components

Concurrency

- With concurrency, services and applications can be shared by clients in a distributed system.

- For an object to be safe in a concurrent environment, its operations must be synchronized in such a way that its data remains consistent.

Concurrency

- Concurrency can be achieved by standard techniques such as semaphores, which are used in most operating systems.

Transparency

- Transparency is defined as the hiding of the separation of components in a distributed systems from the user and the application programmer.

- With transparency the system is perceived as a whole rather than a collection of independent components.

Transparency

- Forms of transparencies:
  - Access transparency
    - Enables local and remote resources to be accessed using identical operations.
  - Location transparency
    - Enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address).

Transparency

- Concurrency transparency
  - Enables several processes to operate concurrently using shared resources without interference between them.
- Replication transparency
  - Enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.

Transparency

- 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.

Transparency

- 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.

Transparency

- The two most important transparencies are access and location transparency referred to together as network transparency.

- Presence or absence of network transparency most strongly affects the utilization of distributed resources.

Question 1.10

- The INFO service manages a potentially very large set of resources, each of which can be accessed by users throughout the Internet by means of a key (a string name). Discuss an approach to the design of the names of the resources that achieves the minimum loss of performance as the number of resources in the service increases. Suggest how the INFO service can be implemented so as to avoid performance bottlenecks when the number of users becomes very large