Goals of Distributed Systems

• Transparency
• Openness
• Scalability

Transparency

• **Transparency** is the concealment from the users and the application programmers of the fact that the processes and resources of a distributed system are physically distributed across multiple computers
• A transparent system is perceived as a whole rather than as a collection of independent components
Types of Transparency (1)

• **Access transparency** enables local and remote resources to be accessed using identical operations
  – E.g., an API for files that uses the same operations to access both local and remote files
• **Location transparency** enables resources to be accessed without knowledge of their physical location
  – Resources are referred by location transparent logical names that contain no information about the physical location of the resource
  – E.g., URLs of Web pages are location transparent
• **Migration transparency** enables resources to be moved without affecting how they can be accessed
  – E.g., a Web page can be moved to a different location without having its URL changed
• **Relocation transparency** enables resources to move while in use without being noticed by users and applications
  – E.g., mobile users can continue to use their laptops while moving from place to place without being disconnected from the Internet

Types of Transparency (2)

• **Concurrent transparency** enables users and applications to access shared resources without interference between each other
  – Concurrent access to a shared resource should leave that resource in a consistent state
  – Consistency can be achieved using locks or transactions
• **Replication transparency** enables multiple instances of resources to be used to increase availability and performance without knowledge of the replicas by users
  – E.g., replicated web contents
• **Failure transparency** enables users and application programs to complete their tasks despite the failure of hardware or software components
  – E.g., email delivery
Openness

- An **open distributed system** is a system that offers services according to **published** standards that describe the syntax and semantics of those services
  - E.g., Internet is an open system as the specifications of Internet protocols are published in RFCs
- Services in distributed systems are generally specified through **interfaces**, which are often described in an **Interface Definition Language** (IDL)
  - Interface definitions written in an IDL specify the **syntax** of the services (i.e., the names of the functions that are available, the types of the parameters, return values, and possible exceptions that can be raised)
  - **Semantics** of interfaces are specified in an informal way by means of natural language

Benefits of Open Distributed Systems

- **Interoperability**: components written by different programmers can easily work together
- **Portability**: applications can be easily ported between different distributed systems that implement the same interfaces
- **Extensibility**: new services can be easily added and old services can be easily re-implemented
Scalability

- A system is said to be **scalable** if it will remain effective when there is a significant increase in the number of users and the number of resources.
- Scalability problems
  - **Size scalability**: as the number of users and resources increase, the system may become overloaded.
  - **Geographical scalability**: as the distance between nodes increases, communication delay becomes significant.

Scaling Techniques

- **Decentralization**
  - achieves size scalability
- **Reducing communication**
  - achieves geographical scalability
- **Replication**
  - achieves size scalability and geographical scalability
Achieving Size Scalability

• To achieve size scalability, we should eliminate performance bottlenecks, including
  – Centralized services (e.g., a single server)
  – Centralized data (e.g., a single DNS table)
  – Centralized algorithms (e.g., routing based on complete information)

Decentralizing Services and Data

• Spreading data and services across multiple machines
• Examples
  – The Web is physically distributed across a large number of Web servers, each handling a collection of Web documents
  – The naming service of DNS is distributed across many name servers
    • The table that maps host names to IP addresses is partitioned between the name servers
Decentralizing Algorithms

• Decentralized algorithms should be used to avoid performance bottlenecks
• Characteristics of decentralized algorithms
  – No machine has complete information about the system state
  – Machines make decisions based only on local information
  – Failure of one machine does not ruin the algorithm
  – No assumption of a global clock
    • It is impossible to get all the clocks in a distributed system exactly synchronized

Reducing Communication

• To achieve geographical scalability, we can reduce communication by moving part of the computation from server to client

The difference between letting (a) a server or (b) a client check forms as they are being filled.
Replication

• When a distributed system grows in size and in geographical coverage, the performance can decrease
• Replicating data and services across a distributed system can improve performance
  – Replicating services balance the load between servers
  – Placing a copy of data near a client reduces communication latency
• Replication lead to consistency problem – modifying one copy makes it different from other copies