Lecture 1

Introduction to Distributed Systems

Presentation Outline

- Introduction
- Defining Distributed Systems
- Characteristics of Distributed Systems
- Example Distributed Systems
- Challenges of Distributed Systems
- Summary

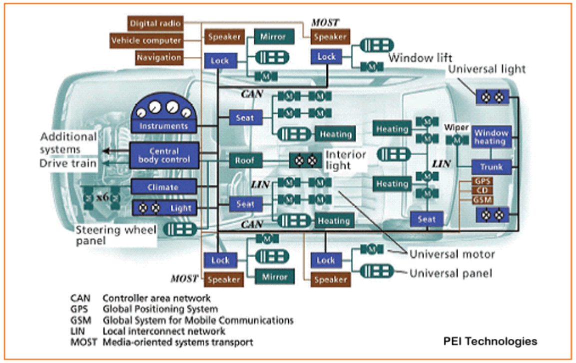
Aims of this module

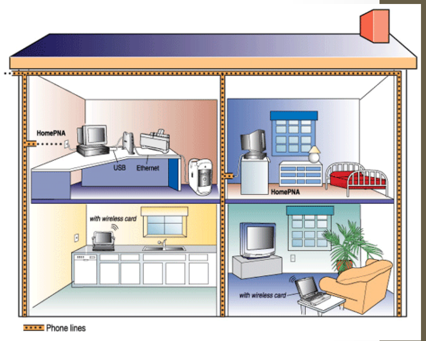
- Introduce the features of Distributed Systems that impact system designers and implementers

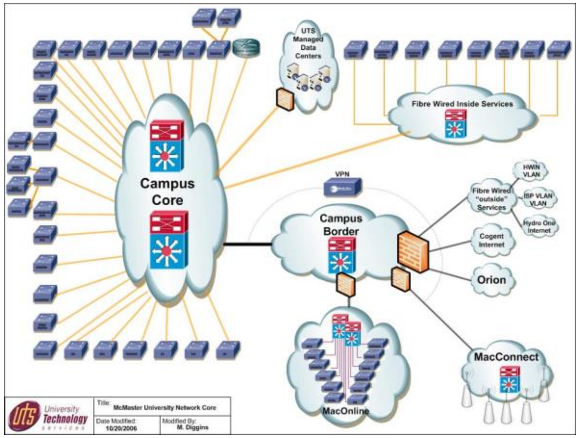
- Introduce the main concepts and techniques that have been developed to help in the tasks of designing and implementing Distributed Systems

Introduction

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







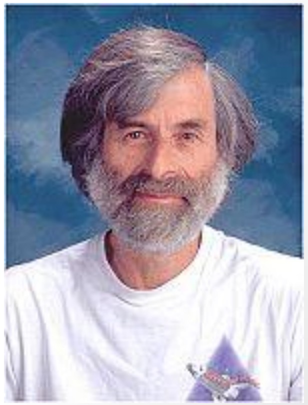
Defining Distributed Systems

- *"A system in which hardware or software components located at networked computers communicate and coordinate their actions only by message passing."* [Coulouris]

- *"A distributed system is a collection of independent computers that appear to the users of the system as a single computer."* [Tanenbaum]

Leslie Lamport's Definition

- "*A distributed system is one on which I cannot get any work done because some machine I have never heard of has crashed.*"
  - Leslie Lamport – a famous researcher on timing, message ordering, and clock synchronization in distributed systems.

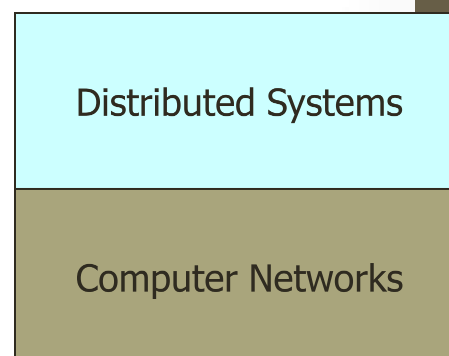


Networks vs. Distributed Systems

- Networks: A media for interconnecting local and wide area computers and exchange messages based on protocols. Network entities are visible and they are explicitly addressed (IP address).

- Distributed System: existence of multiple autonomous computers is transparent

- However,
  - many problems (e.g., openness, reliability) in common, but at different levels.
    - Networks focuses on packets, routing, etc., whereas distributed systems focus on applications.
    - Every distributed system relies on services provided by a computer network.



Reasons for having Distributed Systems

- Functional Separation:
  - Existence of computers with different capabilities and purposes:
    - Clients and Servers

      Clients request services (e.g., a browser accessing a website), while servers process and respond.
    - Data collection and data processing

      IoT devices collect data, which is then processed in a cloud-based distributed system.
- Inherent distribution:
  - Information:

    Different people create and maintain data (e.g., websites, shared documents). Example: Web pages are hosted on different servers worldwide.
    - Different information is created and maintained by different people (e.g., Web pages)
  - People

    Supports remote work, engineering projects, and real-time collaboration. Example: Virtual teams, online meetings, and remote surgery.
    - Computer supported collaborative work (virtual teams, engineering, virtual surgery)
  - Retail store and inventory systems for supermarket chains)

    Large chains (e.g., Walmart) have distributed inventory systems managing stock across multiple locations.

Reasons for having Distributed Systems

- Power imbalance and load variation:
  - Distribute computational load among different computers.

Computational tasks are distributed among different computers to optimize resource usage.
Prevents bottlenecks by balancing workloads across multiple servers.

- Reliability:

Even if one node fails, the system remains operational.
Example: Google Drive, Dropbox store data across multiple servers to prevent data loss.

  - Long term preservation and data backup (replication) at different locations.
- Economies:
  - Sharing resources to reduce costs and maximize utilization (e.g. network printer)
  - Building a supercomputer out of a network of computers.

Reduces costs by sharing hardware and software resources.
Examples:Network printers shared among multiple users instead of separate printers for each.
Grid computing allows multiple computers to work together like a supercomputer (e.g., SETI@home, Folding@home).

Characteristics of Distributed Systems

- Concurrency
  - Carry out tasks independently and parallelely
  - Tasks coordinate their actions by exchanging messages

Different components perform tasks simultaneously and independently.
Tasks communicate via message passing (e.g., REST APIs, gRPC) rather than shared memory.

- Communication via message passing
  - No shared memory

Components interact only through messages, not through shared memory.
Example: Microservices architecture where services communicate via APIs instead of directly sharing variables.

- Resource sharing
  - Printer, database, other services

Enables multiple users to access shared resources

- No global state
  - No single process can have knowledge of the current global state of the system

Example: Distributed databases where no single node has all the data, but they work together to respond to queries.

Characteristics of Distributed Systems

- Heterogeneity – Different devices operating together

Different hardware, operating systems, and network protocols work together.
Example: A distributed system may include Windows servers, Linux servers, and mobile devices accessing the same cloud service.

- Independent and distributed failures

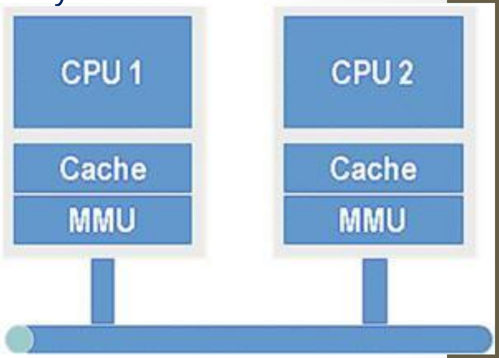
Since components are autonomous, failures can occur independently in different nodes.
Systems must handle fault tolerance (e.g., retries, backups, failover mechanisms).
Example: If one microservice in an e-commerce website fails, others continue running, and users can still browse products.

- No global clock
  - Only limited precision for processes to synchronize their clocks

Example: In blockchain networks, timestamps can slightly vary across nodes, requiring consensus protocols (e.g., Proof of Work).

Differentiation with parallel systems

While distributed systems and parallel systems both involve multiple processors, they differ in architecture, communication, and memory access



Types of Parallel Systems:

- Multiprocessor/Multicore systems
  - Shared memory
  - Bus-based interconnection network
  - E.g. SMPs (symmetric multiprocessors) with two or more CPUs, GPUs
- Multicomputer systems / Clusters
  - No shared memory each computer has its own local memory.

    E.g., Supercomputing clusters like Beowulf clusters.
  - Homogeneous in hard- and software
    - Massively Parallel Processors (MPP)
      - Tightly coupled high-speed network

        Used in large-scale simulations and AI training.
    - PC/Workstation clusters
      - High-speed networks/switches-based connection.

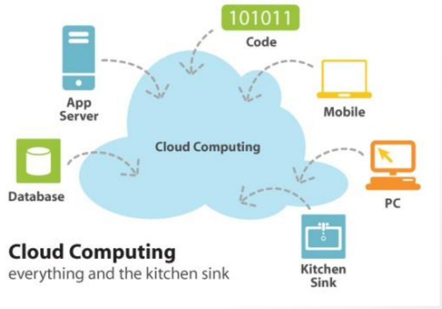
Differentiation with parallel systems is blurring

- Extensibility of clusters leads to heterogeneity
  - Adding additional nodes as requirements grow
- Leading to the rapid convergence of various concepts of parallel and distributed systems

Feature	Parallel Systems	Distributed Systems
Memory	Shared memory (except clusters)	No shared memory
Communication	Bus-based interconnection	Message passing
Coupling	Tightly coupled	Loosely coupled
Scalability	Limited by hardware	Highly scalable
Fault Tolerance	Low (one failure can crash the system)	High (redundancy, failover mechanisms)
Example	Supercomputers, AI Training Clusters	Cloud computing, blockchain, microservices

Examples of Distributed Systems

- They (DS) are based on familiar and widely used computer networks:
  - Internet
  - Intranets, and
  - Wireless networks
- Example DS:
  - Web (and many of its applications like Facebook)
  - Data Centers and Clouds
  - Mobile applicaations
  - Wide area storage systems
  - Banking Systems



Challenges with Distributed Systems

- Heterogeneity
  - Heterogeneous components must be able to interoperate
- Distribution transparency
  - Distribution should be hidden from the user as much as possible
- Fault tolerance
  - Failure of a component (partial failure) should not result in failure of the whole system
- Scalability
  - System should work efficiently with an increasing number of users
  - System performance should increase with inclusion of additional resources

Challenges with Distributed Systems

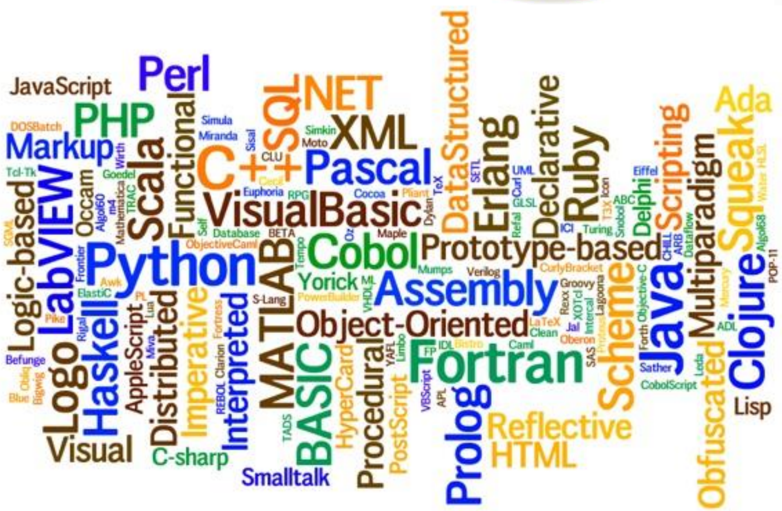
- Concurrency
  - Shared access to resources must be possible
- Openness
  - Interfaces should be publicly available to ease inclusion of new components
- Security
  - The system should only be used in the way intended

Heterogeneity

A distributed system consists of diverse components that must work together

- Heterogeneous components must be able to interoperate across different:
  - Operating systems
  - Hardware architectures
  - Communication architectures
  - Programming languages
  - Software interfaces
  - Security measures
  - Information representation

Example: A cloud-based service might use different database types (SQL & NoSQL) and multiple microservices written in different languages.



Distribution Transparency

- To hide from the user and the application programmer the separation/distribution of components, so that the system is perceived as a whole rather than a collection of independent components.

- ISO Reference Model for Open Distributed Processing (ODP) identifies the following forms of transparencies:

- **Access transparency**
  - Access to local or remote resources is identical
  - E.g. Network File System / **Dropbox**

Accessing Google Drive files without knowing where they are stored.

- **Location transparency**
  - Access without knowledge of location
  - E.g. separation of domain name from machine address.

Users should not need to know the physical location of resources.
Ex- A website hosted on multiple data centers.

Distribution Transparency II

- Failure transparency The system should recover automatically from failures.
  - Tasks can be completed despite failures
  - E.g. message retransmission, failure of a Web server node should not bring down the website
- Replication transparency Users should not notice if data is replicated across multiple locations.

  backup
  - Access to replicated resources as if there was just one. And provide enhanced reliability and performance without knowledge of the replicas by users or application programmers.
- Migration (mobility/relocation) transparency
  - Allow the movement of resources and clients within a system without affecting the operation of users or applications.
  - E.g. switching from one name server to another at runtime; migration of an agent/process from one node to another.

Ex- Moving a virtual machine to another server

Distribution Transparency III

- **Concurrency transparency** Multiple users can access shared resources simultaneously.
  - A process should not notice that there are other sharing the same resources
- **Performance transparency:**
  - Allows the system to be reconfigured to improve performance as loads vary The system adjusts dynamically to workload changes.
  - E.g., dynamic addition/deletion of components, switching from linear structures to hierarchical structures when the number of users increases - Autoscaling in cloud computing.
- **Scaling transparency:**
  - Allows the system and applications to expand in scale without changes in the system structure or the application algorithms.
- **Application level transparencies:**
  - Persistence transparency
    - Masks the deactivation and reactivation of an
  - Transaction transparency
    - Hides the coordination required to satisfy the properties of operations

Type of Transparency	Description	Example
Access Transparency	Access to local and remote resources should be identical.	Accessing Google Drive files without knowing where they are stored.
Location Transparency	Users should not need to know the physical location of resources.	A website hosted on multiple data centers.
Failure Transparency	The system should recover automatically from failures.	Auto-retry mechanism when a request fails.
Replication Transparency	Users should not notice if data is replicated across multiple locations.	Cloud-based file synchronization (e.g., Google Drive, OneDrive).
Migration Transparency	Resources can move without disrupting services.	Moving a virtual machine to another server.
Concurrency Transparency	Multiple users can access shared resources simultaneously.	Database transactions managing concurrent access.
Performance Transparency	The system adjusts dynamically to workload changes.	Autoscaling in cloud computing.
Scaling Transparency	The system grows without requiring structural changes.	Adding servers to a load balancer in a web app.

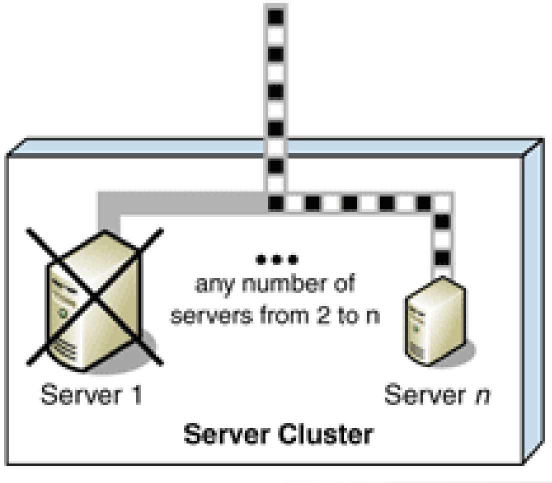
Fault Tolerance

- Failure: an offered service no longer complies with its specification
- Fault: cause of a failure (e.g. crash of a component)

without being affected

- Fault tolerance: no failure despite faults

- **Failure** – A system component stops functioning as expected.
- **Fault** – The cause of a failure (e.g., hardware crash, network failure).
- **Fault Tolerance** – The system should continue functioning despite faults.



Fault Tolerance Mechanisms

- Fault detection
  - Checksums, heartbeat, …
- Fault masking
  - Retransmission of corrupted messages, redundancy, …
- Fault toleration
  - Exception handling, timeouts,…
- Fault recovery
  - Rollback mechanisms,…

Mechanism	Description	Example
Fault Detection	Identifying issues before they cause failures.	Heartbeat signals, checksums.
Fault Masking	Hiding faults so they do not affect the system.	Retransmission of corrupted messages.
Fault Tolerance	Handling faults dynamically.	Load balancing, exception handling.
Fault Recovery	Restoring the system to a previous state.	Database rollback, failover servers.

Scalability

Distributed systems must handle increasing numbers of users and resources efficiently.

- System should work efficiently at many different scales, ranging from a small Intranet to the Internet
- Remains effective when there is a significant increase in the number of resources and the number of users
- Challenges of designing scalable distributed systems:
  - Cost of physical resources
  - Performance Loss
  - Preventing software resources running out:
    - Numbers used to represent Internet addresses (32 bit->64bit)
    - Y2K-like problems
  - Avoiding performance bottlenecks:
    - Use of decentralized algorithms (centralized DNS to decentralized)

Concurrency

Multiple users or processes access shared resources at the same time.

- Provide and manage concurrent access to shared resources:
  - Fair scheduling
  - Preserve dependencies (e.g. distributed transactions)
  - Avoid deadlocks
  - Preserve integrity of the system

Openness and Interoperability

- Open system:
  "... a system that implements sufficient open specifications for interfaces, services, and supporting formats to enable properly engineered applications software to be ported across a wide range of systems with minimal changes, to interoperate with other applications on local and remote systems, and to interact with users in a style which facilitates user portability" (Guide to the POSIX Open Systems Environment, IEEE POSIX 1003.0)

Openness and Interoperability

- Open message formats: e.g. XML
- Open communication protocols: e.g. HTTP, HTTPS
- Open spec/standard developers - communities:
  - ANSI, IETF, W3C, ISO, IEEE, OMG, Trade associations,...

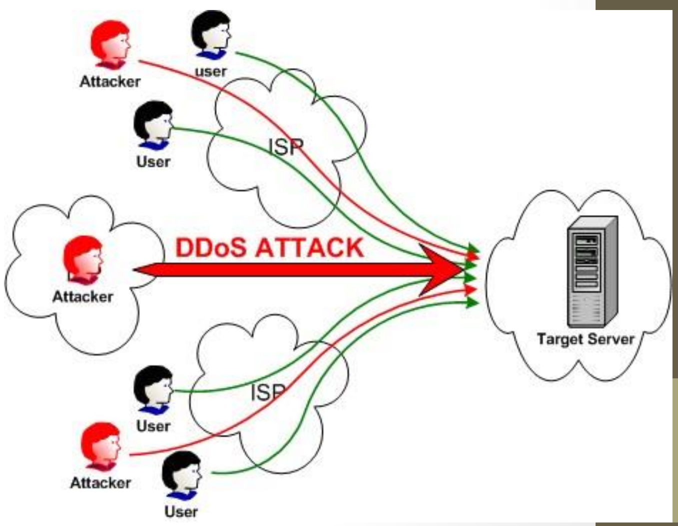
Open message formats (e.g., XML, JSON)
Open communication protocols (e.g., HTTP, HTTPS)
Standardization bodies (ISO, IEEE, W3C)

Security I

- Resources are accessible to authorized users and used in the way they are intended
- Confidentiality
  - Protection against disclosure to unauthorized individual information
  - E.g. ACLs (access control lists) to provide authorized access to information
- Integrity
  - Protection against alteration or corruption
  - E.g. changing the account number or amount value in a money order

Security II

- Availability
  - Protection against interference targeting access to the resources.
  - E.g. denial of service (DoS, DDoS) attacks
- Non-repudiation
  - Proof of sending / receiving an information
  - E.g. digital signature



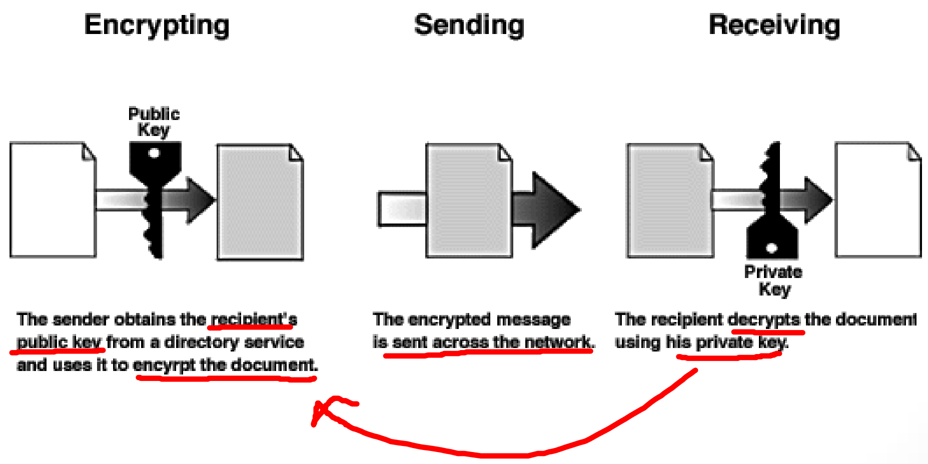
Key Security Concepts:

Security Feature	Description	Example
Confidentiality	Prevents unauthorized access.	Encryption, ACLs.
Integrity	Protects against data corruption.	Digital signatures, hashing.
Availability	Ensures system resources remain accessible.	Protection against DoS/DDoS attacks.
Non-repudiation	Prevents users from denying actions.	Digital certificates, blockchain transactions.

Security Mechanisms

- Encryption - Protects data
  - E.g. Blowfish, RSA
- Authentication - Confirms user identity
  - E.g. password, public key authentication
- Authorization - Controls access
  - E.g. access control lists

    role-based access control



Business Example and Challenges

- Web/Mobile app to search and purchase online courses
  - Customers can connect their computer to your server (locally hosted or cloud hosted):
    - Browse your courses
    - Purchase courses
    - …

This example has been adapted from Torbin Weis, Berlin University of Technology

Business Example – Challenges I

- What if
  - Your customers use different devices? (Dell laptop, Android device …)
  - Your customers use different OSs? (Android, IoS, Ubuntu…)
  - … … a different way of representing data? (Text, Binary,…)
  - **Heterogeneity**
- Or
  - You want to move your business and computers to China (because of the lower costs)?
  - Your client moves to a different country(more likely)?
  - **Distribution transparency**

Business Example – Challenges II

- What if
  - Two customers want to order the same item at the same time?
  - **Concurrency**
- Or
  - The database with your inventory information crashes?
  - Your customer's computer crashes in the middle of an order?
  - **Fault tolerance**

Business Example – Challenges III

- What if
  - Someone tries to break into your system to alter data?
  - … sniffs for information?
  - … someone says they have enrolled to the course but they haven’t?
  - **Security**
- Or
  - You are so successful that millions of people are using your app at the same time.
  - **Scalability**

Business Example – Challenges IV

- When building the system…
  - Do you want to write the whole software on your own (network, database,…)?
  - What about updates, new technologies?
  - Adding a web client later on?
  - Will your system need to communicate with existing systems (e.g. payment gateways, SMS servers)
  - **Reuse** and **Openness** (Standards)

Impact of Distributed Systems

- New business models (e.g. Uber, Airbnb)
- Global financial markets
- Global labor markets
- E-government (decentralized administration)
- Ecommerce
- Driving force behind globalization
- Social/Cultural impact
- Media getting decentralized

Summary

- Distributed Systems are everywhere
- The Internet enables users throughout the world to access its services wherever they are located
- Resource sharing is the main motivating factor for constructing distributed systems
- Construction of DS produces many challenges:
  - Heterogeneity, Openness, Security, Scalability, Failure handling, Concurrency, and Transparency
- Distributed systems enable globalization:
  - Community (Virtual teams, organizations, social networks)
  - Science (e-Science)
  - Business (e-Bussiness)