CSE 601: Distributed Systems

Toukir Ahammed

Course Overview

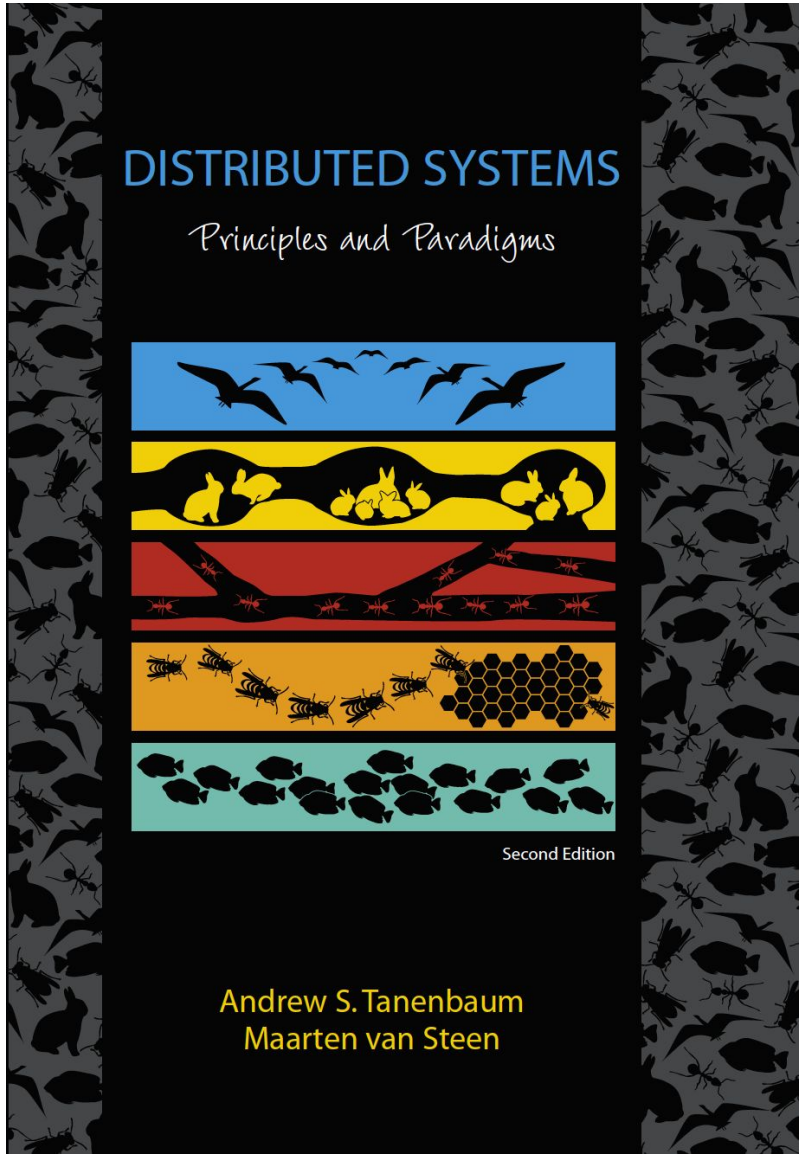
Course Title: Distributed Systems

Course Code: CSE601

Credit: 3 Credits (2 Credit Theory + 1 Credit Lab)

Reference Book:

Distributed systems: Principles and Paradigms by Andrew S. Tanenbaum (*2nd edition*)



Marks Distribution

- Attendance 5
- Midterm 15
- Presentation 5
- Lab/Project 25
- Assignment 20
- Final Exam 30

Google Classroom

p44adob

Introduction to Distributed System

Definition of a Distributed System

A distributed system is:

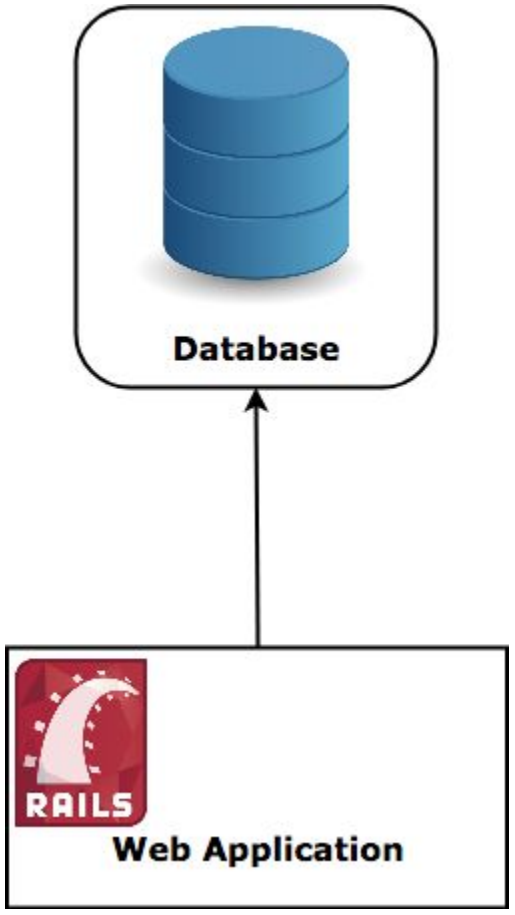
"A collection of independent computers that appears to its users as a single coherent system."

"A computing environment in which various components are spread across multiple nodes (*computer, phone, car, robot or other computing devices*) on a network trying to achieve some task together."

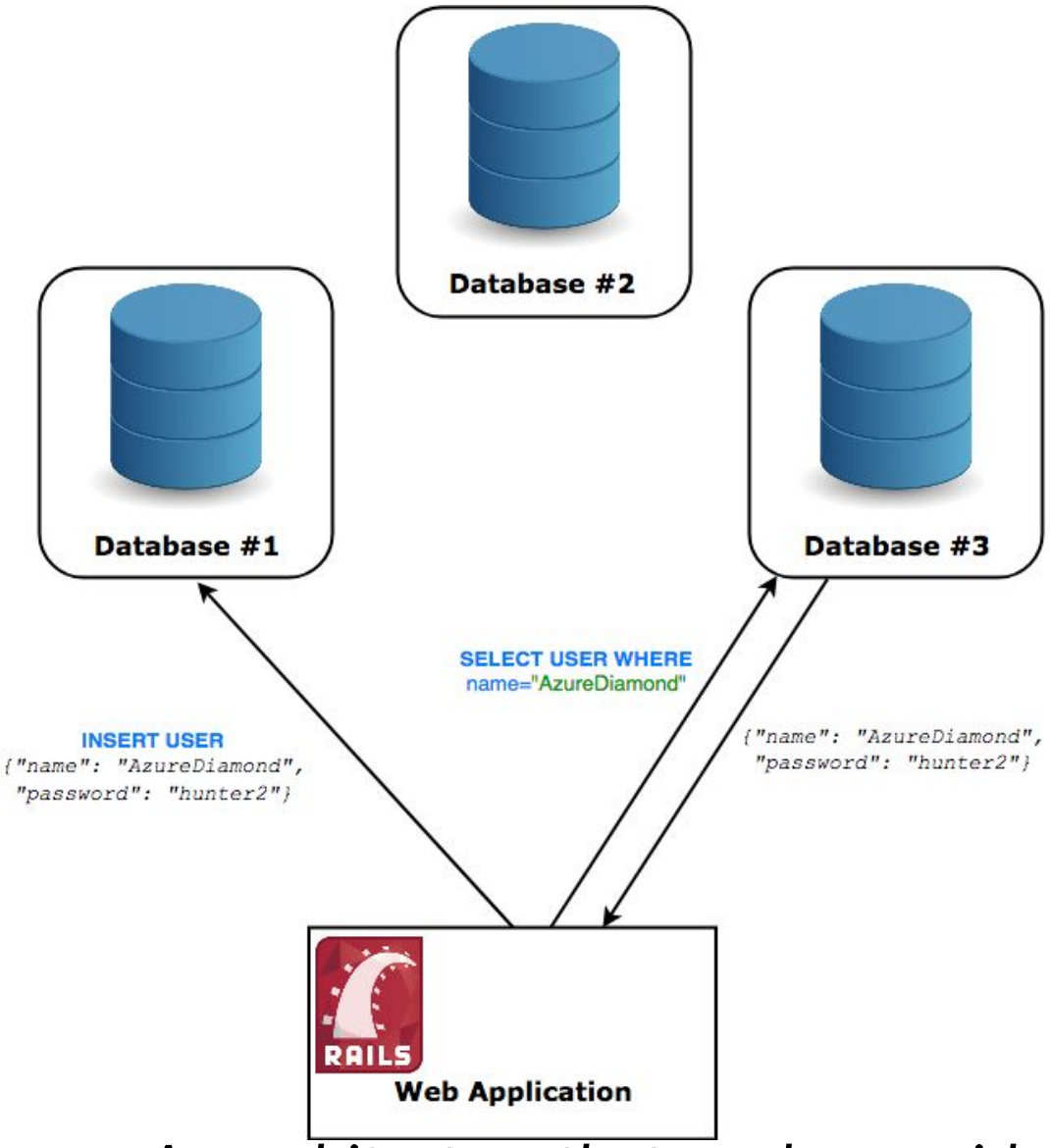
Examples of distributed systems

- Machine learning (for compute)
- P2P file sharing (high availability, share large files, piracy)
- Google search engine (for storage and bandwidth)
- Facebook (for storage and bandwidth)
- Black hole image (distributed observation)
- IOT (Sensors on a network)
- Blockchain (decentralized record of transactions) etc.

Example

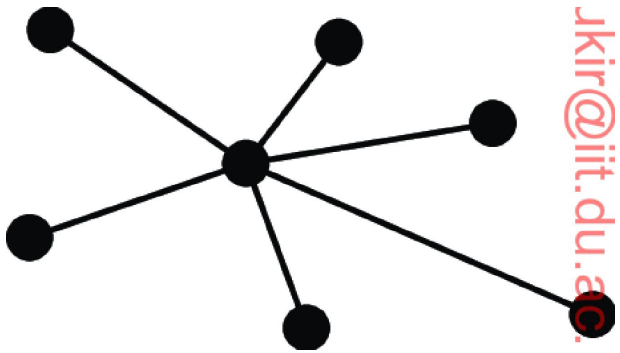


A traditional stack

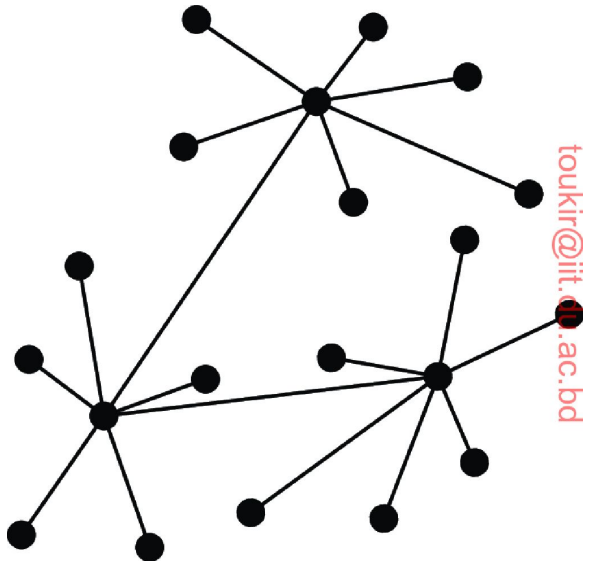


An architecture that can be considered distributed

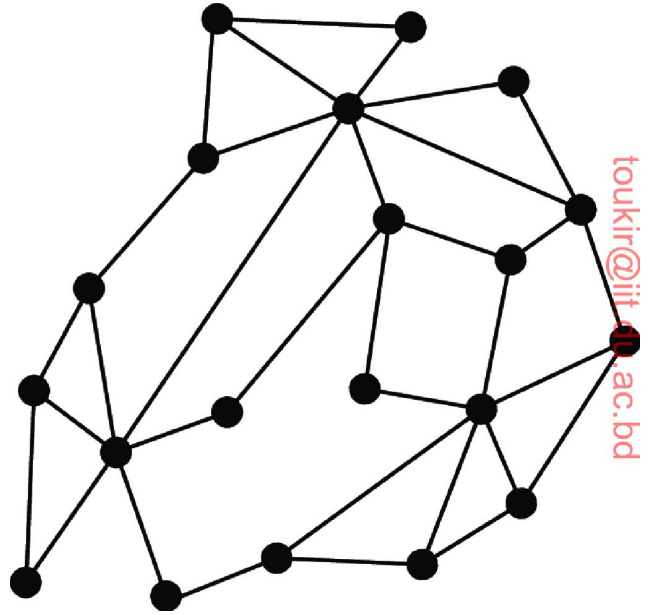
Distributed versus decentralized systems



centralized



decentralized



distributed

Distributed versus decentralized systems

- **Decentralized** is still distributed in the technical sense, but the whole decentralized systems is not owned by one actor. No one company can own a decentralized system, otherwise it wouldn’t be decentralized anymore.

- This means that most systems we will go over today can be thought of as **distributed centralized systems** — and that is what they’re made to be.

Why make a system distributed?

- It's inherently distributed:
  e.g. sending a message from your mobile phone to your friend's phone
- For better reliability:
  even if one node fails, the system as a whole keeps functioning
- For better performance:
  - get data from a nearby node rather than one halfway round the world
- To solve bigger problems:
  - e.g. huge amounts of data, can't fit on one machine

Advantages of Distributed Systems

- **Scalability**: Distributed systems are made on default to be scalable. Whenever there is an increase in workload, users can add more workstations. There is no need to upgrade a single system. Moreover, no any restrictions are placed on the number of machines.

- **Reliability**: Distributed systems are far more reliable than single systems in terms of failures. Even in the case of a single node malfunctioning, it does not pose problems to the remaining servers. Other nodes can continue to function fine.

Advantages of Distributed Systems

- **Low Latency**: Since users can have a node in multiple geographical locations, distributed systems allow the traffic to hit a node that's closest, resulting in low latency and better performance.

- **Efficiency**: Distributed systems allow breaking complex problems/data into smaller pieces and have multiple computers work on them in parallel, which can help cut down on the time needed to solve/compute those problems.

Why NOT make a system distributed?

The trouble with distributed systems:

- Communication may fail (and we might not even know it has failed).
- Processes may crash (and we might not know).
- All of this may happen non-deterministically.

Fault tolerance: we want the system as a whole to continue working, even when some parts are faulty.

- This is hard.
- Writing a program to run on a single computer is comparatively easy?!

Challenges of Distributed Systems

- **Heterogeneity** (variety and difference): the differences that arise in networks, programming languages, hardware, operating systems and differences in software implementation.
- **Openness**: the system's extensibility and ability to be reimplemented. It can be measured by 3 characteristics:
  - **interoperability**: system's ability to effectively interchange information between computers by standardization
  - **portability**: system's ability to properly function on different operating systems
  - **extensibility**: allowing developers to freely add new features or easily reimplement existing ones without impairing functionality

Challenges of Distributed Systems

- **Security**: consists of 3 components
  - confidentiality: protection against disclosure to unauthorized individuals
  - integrity: protection against alteration or corruption
  - availability: protection against interference with the means to access the resources.

- **Concurrency**: multiple clients can access a shared resource at the same time. Thus, steps need to be taken to ensure that any manipulation in the system remains in a stable state. However, the illusion of simultaneous execution should be preserved.

Challenges of Distributed Systems

- **Scalability**: the measure of a system's ability to increase or decrease in performance and cost in response to changes in application and system processing demands.
  - Example: how well a hardware system performs when the number of users is increased, how well a database withstands growing numbers of queries, or how well an operating system performs on different classes of hardware.
- **Failure handling**: When faults occur, programs may produce incorrect results or may stop before they have completed the intended computation. Any process, computer or network may fail independently of the others. Therefore each component needs to be aware of the possible ways in which the components it depends on may fail and be designed to deal with each of those failures appropriately.

Challenges of Distributed Systems

- **Transparency**: system that is able to present itself to users and applications as if it were only a single computer system is said to be transparent.

Transparency	**Description**
Access	Hide differences in data representation and how an object is accessed
Location	Hide where an object is located
Relocation	Hide that an object may be moved to another location while in use
Migration	Hide that an object may move to another location
Replication	Hide that an object is replicated
Concurrency	Hide that an object may be shared by several independent users
Failure	Hide the failure and recovery of an object

Design Goals of a Distributed Systems

Resource Sharing

Making Resource Accessible

- An important goal of a distributed system is to make it easy for users (and applications) to access and share remote resources.
- *Example*: printers, computers, storage facilities, data, files, services, networks etc.
- Economic reason: it makes economic sense to share costly resources such as supercomputers, high-performance storage systems, e.g., it is cheaper to
  - have a single high-end reliable storage facility be shared than having to buy and maintain storage for each user separately.
  - it is cheaper to let a printer be shared by several users in an office than having to buy and maintain a separate printer for each user

Making Resource Accessible

- More advantages
  - Connecting users and resources also makes it easier to collaborate and exchange information
  - collaboration tools such as software for collaborative editing, teleconferencing, and so on that became (more easily) available due to the COVID-19 pandemic

Design Goals of a Distributed Systems

Distribution transparency

Distribution transparency

- An important goal of a distributed system is to hide the fact that its processes and resources are physically distributed across multiple computers, possibly separated by large distances.
- In other words, it tries to make the distribution of processes and resources **transparent**, that is, invisible, to end users and applications.

Types of distribution transparency

Transparency	Description
Access	Hide differences in data representation and how an object is accessed
Location	Hide where an object is located
Relocation	Hide that an object may be moved to another location while in use
Migration	Hide that an object may move to another location
Replication	Hide that an object is replicated
Concurrency	Hide that an object may be shared by several independent users
Failure	Hide the failure and recovery of an object

Access Transparency

Access transparency deals with hiding differences in data representation and the way that objects can be accessed.

Example:

- A distributed system may have computer systems that run different operating systems, each having their own file-naming conventions.
- Access issues that should preferably be hidden:
  - differences in naming conventions
  - differences in file operations, or
  - differences in low-level communication with other processes

Location Transparency

- **Location transparency** refers to the fact that users cannot tell where an object is physically located in the system.
- **Naming** plays an important role in achieving location transparency. It can often be achieved by assigning only logical names to resources, i.e., names in which the location of a resource is not secretly encoded.
- **Example**: **uniform resource locator** (**URL**) such as https://www.distributed-systems.net/, gives no clue about the actual location of the Web server

Migration Transparency

- Distributed systems in which resources can be moved without affecting how those resources can be accessed are said to provide **migration transparency**.
- In the previous example, the URL also gives no clue whether files at that site have always been at their current location or were recently moved there.

Relocation Transparency

- **Relocation transparency** supports the mobility of processes and resources initiated by users, without affecting ongoing communication and operations. In such cases, resources can be relocated while they are being accessed without the user or application noticing anything
- **Example**:
  - communication between mobile phones: regardless whether two people are actually moving, mobile phones will allow them to continue their conversation.
  - online tracking and tracing of goods as they are being transported from one place to another

Replication Transparency

- In distributed systems, resources may be replicated to increase availability or to improve performance by placing a copy close to the place where it is accessed.
- **Replication transparency** deals with hiding the fact that several copies of a resource exist, or that several processes are operating so that one can take over when another fails.
- *Note*: a system that supports replication transparency should generally support location transparency as well, because it would otherwise be impossible to refer to replicas at different locations.

Concurrency Transparency

- An important goal of distributed systems is to allow sharing of resources. In many cases, sharing resources is done *cooperatively*. However, there are also many examples of *competitive* sharing of resources.
- **Example**: two independent users may each have stored their files on the same file server or may be accessing the same tables in a shared database. In such cases, it is important that each user does not notice that the other is making use of the same resource. This phenomenon is called **concurrency transparency**.
- An important issue of concurrent access is the consistency of the resource. It can be achieved through locking mechanisms, by which users are given exclusive access to the desired resource.

Failure Transparency

- **Failure transparency** means that a user or application does not notice that some piece of the system fails to work properly, and that the system subsequently (and automatically) recovers from that failure.
- Masking failures is one of the hardest issues in distributed systems and is even impossible in certain cases. The main difficulty in masking and transparently recovering from failures lies in the inability to distinguish between a dead process and a painfully slowly responding one.
- **Example**: when contacting a busy Web server, a browser will eventually time out and report that the Web page is unavailable. At that point, the user cannot tell whether the server is actually down or that the network is badly congested.

Design Goals of a Distributed Systems

Openness

Openness

- Another important goal of distributed systems is openness.
- An **open distributed system** is essentially a system that offers components that can easily be used by, or integrated into other systems.
- At the same time, an open distributed system itself will often consist of components that originate from elsewhere.

Openness: Interoperability

- **Interoperability** characterizes the extent by which two implementations of systems or components from different manufacturers can co-exist and work together by merely relying on each other's services as specified by a common standard.
- **Example**: a healthcare system where hospitals and clinics, using different software platforms, exchange patient data seamlessly through open standards

Openness: Portability

- **Portability** characterizes to what extent an application developed for a distributed system A can be executed, without modification, on a different distributed system B that implements the same interfaces as A.
- **Example**: cloud-based application that can be easily moved from one cloud provider, like AWS, to another, such as Google Cloud, without significant modification (containerization technologies like **Docker)**

Openness: Extensibility

- It should be easy to add new components or replace existing ones without affecting those components that stay in place. In other words, an open distributed system should also be **extensible**.
- In an extensible system, it should be relatively easy to add parts that run on a different operating system, or even to replace an entire file system.
- **Example**: plug-ins for Web browsers, but also those for Websites, such as the ones used for WordPress.

Design Goals of a Distributed Systems

Scalability

Scalability dimensions

- **Size scalability:** A system can be scalable regarding its size, meaning that we can easily add more users and resources to the system without any noticeable loss of performance.
- **Geographical scalability:** A geographically scalable system is one in which the users and resources may lie far apart, but the fact that communication delays may be significant is hardly noticed.
- **Administrative scalability:** An administratively scalable system is one that can still be easily managed even if it spans many independent administrative organizations.

Size Scalability

- **The computational capacity, limited by the CPUs**
- **Example**:
  - A service for computing optimal routes taking real-time traffic information into account
- **Problems**:
  - this may be primarily a *compute-bound* service, requiring several (tens of) seconds to complete a request
  - if there is only a single machine available, then even a modern high-end system will eventually run into problems when the number of requests increases beyond a certain point

Size Scalability

- **The storage capacity, including the I/O transfer rate**
- **Example**:
  - a poorly designed centralized search engine
- **Problems**:
  - need to match a query against an entire data set
  - huge amount of data can exceed the main-memory capacity
  - much of the processing time will be determined by the relatively slow disk accesses and transfer of data between disk and main memory

Size Scalability

- **The network between the user and the centralized service**
- **Example**:
  - a video-on-demand service that needs to stream high-quality video to multiple users
- **Problems**:
  - a video stream can easily require a bandwidth of 8 to 10
  - if a service sets up point-to-point connections with its customers, it may soon hit the limits of the network capacity

Geographical Scalability

- many of distributed systems designed for local-area networks are based on **synchronous communication**
- in a wide-area system, we need to consider that inter-process communication may be hundreds of milliseconds
- **Example**:
- Streaming video: In a home network, even with wireless links, ensuring a stable, fast stream of high-quality video frames from a media server to a display is quite simple.
- Simply placing that same server far away will surely fail: *bandwidth limitations will instantly surface*

Administrative Scalability

- The concern is how to scale a distributed system across multiple, independent administrative domains
- A major problem that needs to be solved is that of conflicting policies regarding resource usage (and payment), management, and security
- a distributed system that reside within a single domain can often be trusted by users
- system administration may have tested and certified applications, and may have taken special measures to ensure that such components cannot be tampered with

Administrative Scalability

- If a distributed system expands to another domain, two types of security measures need to be taken
- has to protect itself against malicious attacks from the new domain
  - Users from the new domain may have only read access to the file system
  - facilities may not be made available to unauthorized users
- has to protect itself against malicious attacks from the distributed system
- **Counterexample**: some distributed systems spanning multiple administrative domains that apparently do not suffer from administrative scalability problems (e.g., modern file-sharing peer-to-peer networks)