CSE 601: Distributed Systems

Toukir Ahammed

Course Overview

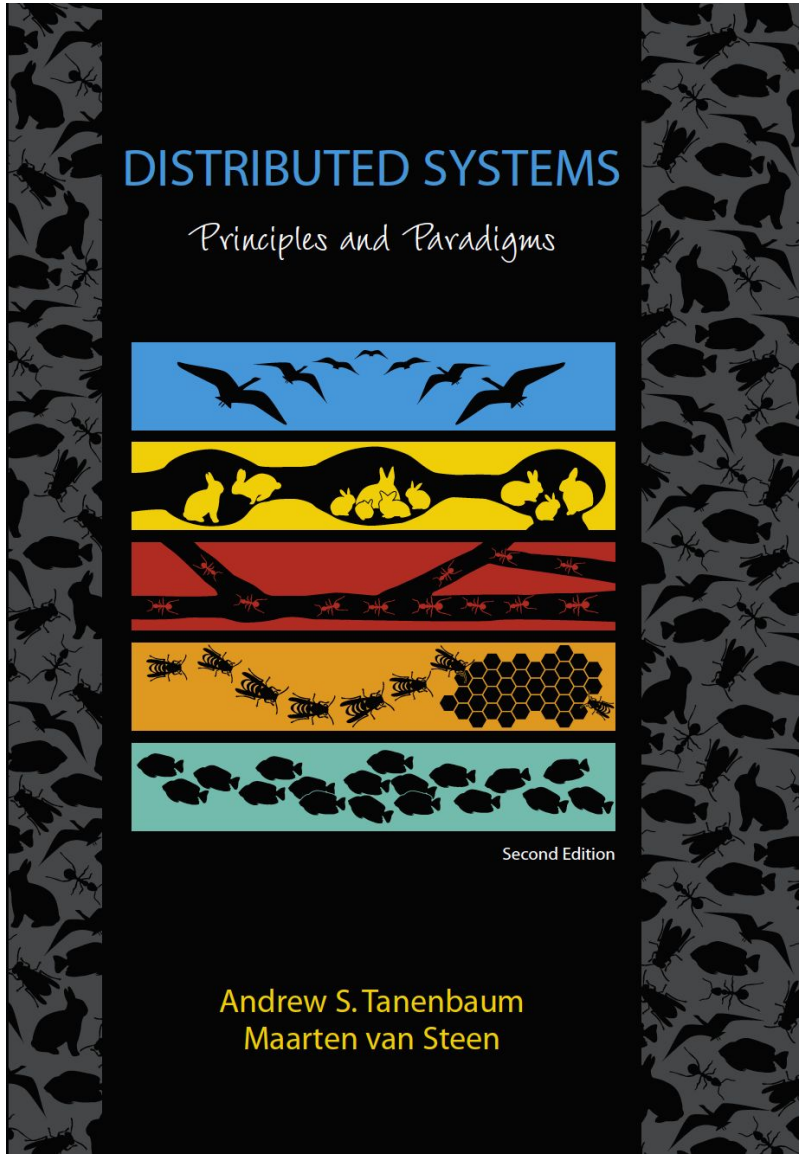
Course Title: Distributed Systems

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:

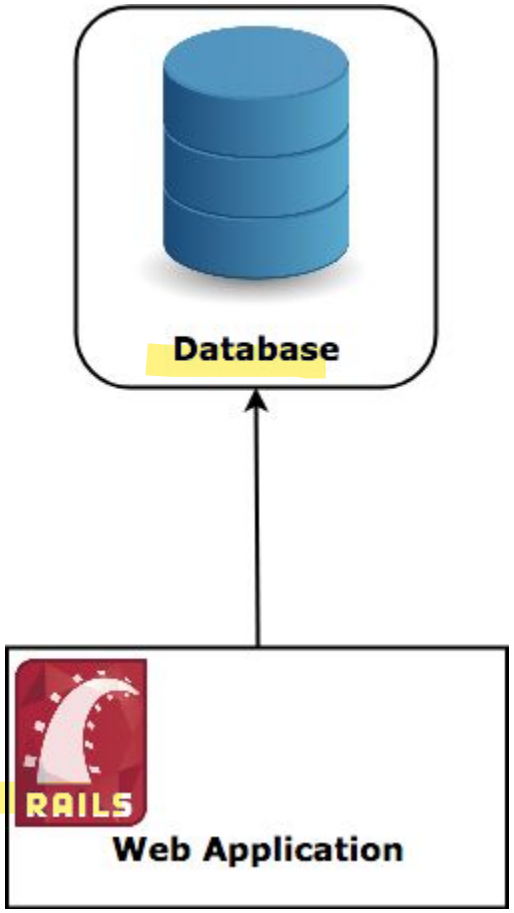
"A collection of independent computers that appears to its users as a single coherent system."

"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."

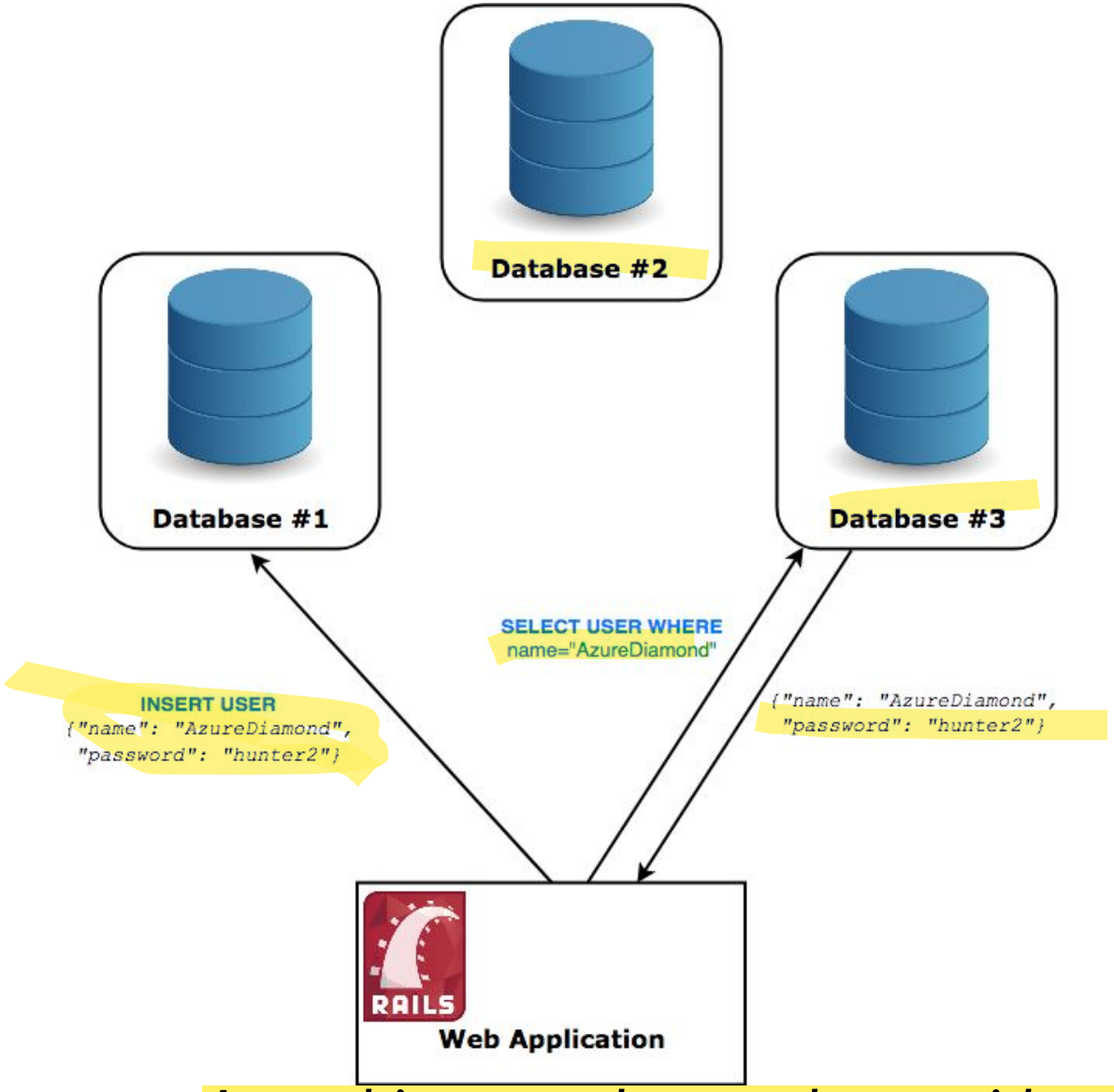
Examples of distributed systems

- 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

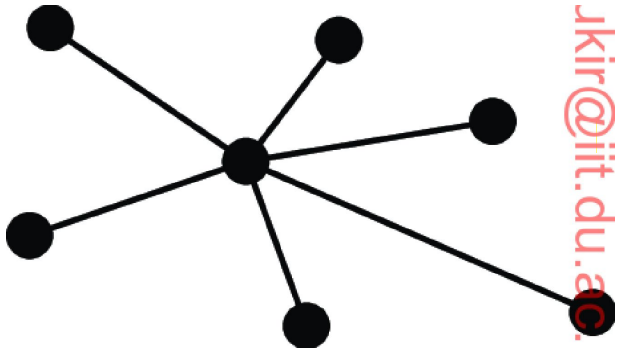


A traditional stack

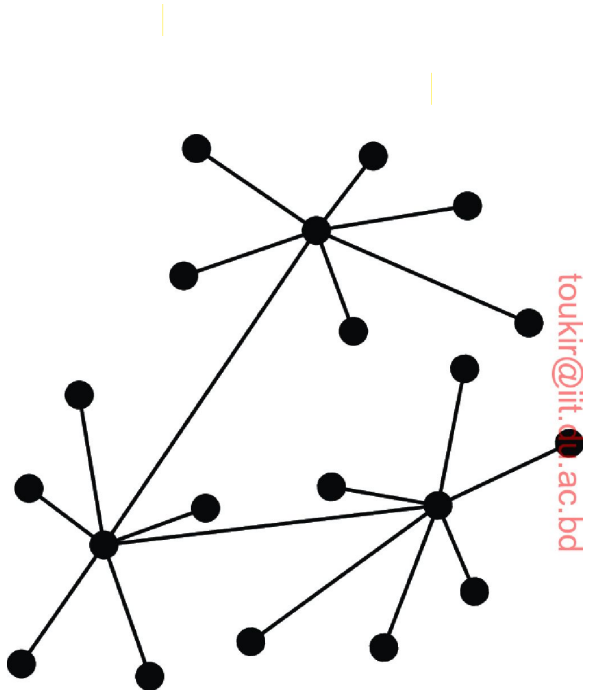


An architecture that can be considered distributed

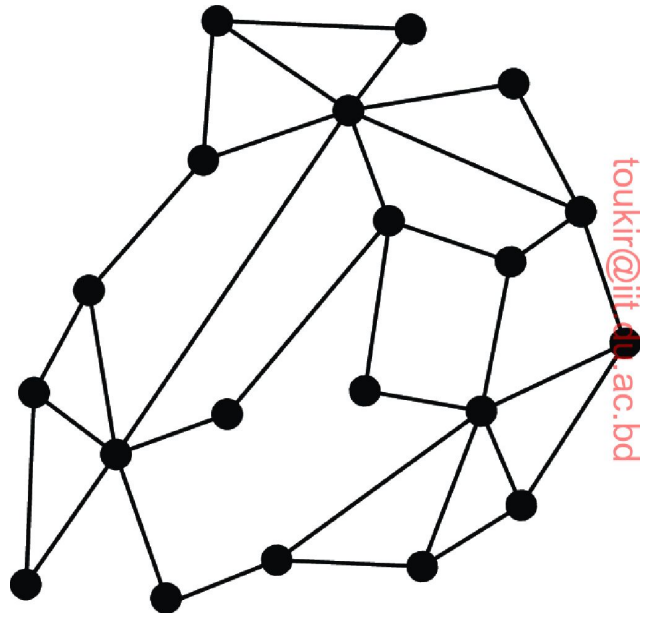
Distributed versus decentralized systems



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