Module 1 Lecture 1: **Introduction to Distributed Systems**
Prepared by Gwyneth Ang for *NTU CE/CZ4153 Intro to Blockchain*

---

## 1. What is a distributed system?

A distributed system is a set of networked computers (also referred to as autonomous nodes or computing elements) that communicate and coordinate their actions by passing messages to address a problem collectively (address a common problem), appearing as a single coherent system to its users.

**Analogy: Soccer**

| Definition | Soccer |
|---|---|
| Communicate and coordinate their actions by passing messages | "Pass me the ball!", hand signals, coded messages, etc. |
| Address a problem collectively | Maximize number of goals while minimizing number of goals opponent scores. |
| Single coherent system | Players play as one team. Although each player has their own role, such as attack or defense, they are one body(system). |

## 2. Fundamental Characteristics of Distributed Systems

1. **Concurrency**
   All components run concurrently; programs execute concurrently.
2. **Loosely Coupled**
   There is no global clock and global shared memory.
3. **Independent Failures**
   Any component can fail at any time and failures occur of each other.

**Analogy: Soccer**

| Characteristics | Soccer |
|---|---|
| Concurrency | Individual players are autonomous, making their own moves at the same time. |
| Loosely Coupled | Each player has their own "clock". E.g., if coach says kick the ball 1 second after whistle is blown, different players count differently, though marginally. No global shared memory: cannot access a central "bank" of commands, players must store it themselves, or use some method to derive command meaning. |
| Independent Failures | Assume failure is equivalent to being out of the game while it is still in play. If one player is injured/ red card(failed), other players in the team will not be out of the game. |

## 3. Architecture and Communication

In terms of system architecture, distributed systems can have *client-server model* or *peer-to-peer model (P2P)*. In the set of networked computers, each node has a role. Generally, all processes in P2P play similar roles, unlike client-server models. Clients use and access services through Servers, while Servers provide services by managing resources by accepting requests from clients and responding to them.

In this course, we focus on the P2P model. P2P systems may be structured, unstructured or hierarchical.

In *Unstructured P2P Systems*, each node maintains an ad hoc list of neighbours. Other nodes cannot anticipate or expect which nodes are neighbours and which node carries the data we are interested in. The only way to find out is to send a message to other nodes and wait for the network to propagate the message until we find the responsible peer.

*Structured P2P Systems* have a specific deterministic topology. Nodes follow a well-defined algorithm to determine what role a node takes on. Take the example of a distributed file system. By maintaining a Distributed Hash Table and using hash functions to assign a key to each node and each data item, each peer is responsible for a specific set of keys. Thus, when a query for a particular file is made, a global protocol is followed to determine which peer is responsible for the which data. This makes the direction of search towards the responsible peer more efficient than in unstructured systems.

In *Hierarchical P2P Systems*, groups are organized in top-level overlay network. Each group has its autonomous intra-group overlay and lookup service. To find the peer responsible for a key, the top-level overlay first determines the group responsible for the key; the responsible group then uses its intra-group overlay to determine the specific peer that is responsible for the key. We use the analogy of a dinner party. Most guests are free to connect with each other, there is no hierarchy or role difference. Most people in the party are equal, except for the receptionist and host, who have dedicated jobs and can be considered as the "leader" among the crowd as they organize the crowd and guide newcomers.

## 4. Consistency and Replication

To promote reliability and scalability, we use replication in distributed systems. Replication refers to organizing data such that each data element is copied and stored in several components within a system.

**Advantages**

1. It makes systems more reliable and fault tolerant as there would be other replicas available should a component fail.
2. Replicas also help with scalability as load access can be balanced, as not all processes accessing the same data is dependent on a single component. Replication is important for performance when distributed systems need to scale in numbers and in geographical locations as it can allow work to be divided, thus sharing the load. For example, it is common for companies to have read replicas of database web services across different regions as they might have users from around the world accessing data in large quantities. Having a copy of the data in the proximity of end users also decreases the time required to access the data.

However, there comes a trade-off between performance and maintaining consistency across replicas. When a particular copy is modified, that copy is different from the rest. To ensure consistency, other copies need to be modified as well. How and when the modifications are carried out affect the price of replication. We determine the "how and when" through the different consistency models, which help us determine the policies and rules to which we carry out updates to other copies. They exist to help multiple nodes achieve a certain agreed upon value through specific rules.

There are different consistency models, such as continuous consistency, sequential consistency, strict consistency and many more. In sequential consistency, maintains the order of the operations occurring across different processes. Let us assume that there are 3 processes in our system, with each process having two operations in the following order:

| Process A | Process B | Process C |
|---|---|---|
| A1 | | C1 |
| A2 | B1 | |
| | B2 | |
| | | C2 |

Which of the following sequences of operations follow Sequential Consistency?

1. A1, A2, B1, B2, C1, C2
2. B1, A1, B2, A2, C1, C2
3. A1, B2, A2, C1, C2, B1
4. A1, B1, C1, C2, B2, A2
5. C1, A1, B1, C2, B2, A2
6. A2, B1, B2, C1, C2, A1

1, 2, 4, and 5 follow Sequential Consistency as the order of operations issued by each process is preserved. First operation of a process must come before the second operation of the same process.