NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Distributed Systems

Introduction to Distributed Systems

WEN Yonggang
Professor, SCSE, NTU

*"A distributed system is a collection of **autonomous computing elements** that appears to its users as a **single coherent system.**"*

Distributed Systems (Third Edition, 2018)

Maarten van Steen and Andrew S Tanenbaum

Datacenters

ATM Network

The Internet

Types of Distributed Systems

High performance distributed computing : **Cluster, Grid, Cloud, Fog**

Examples : Singapore Supercomputer, Amazon EC2 Cloud Service

Distributed information systems : **Ledgers, Databases, Storages**

Examples : Public Blockchain Network, Apache Cassandra Systems

Pervasive systems : **Ubiquitous computing, Mobile computing**

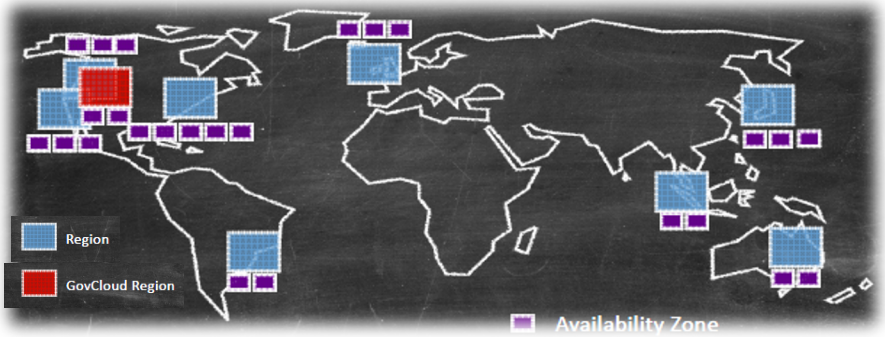
Examples : Self-Driving Vehicular Network, Wearable Devices

Distributed Computing System – Amazon Cloud

AWS is a collection of remote computing services that together make up a cloud computing platform.

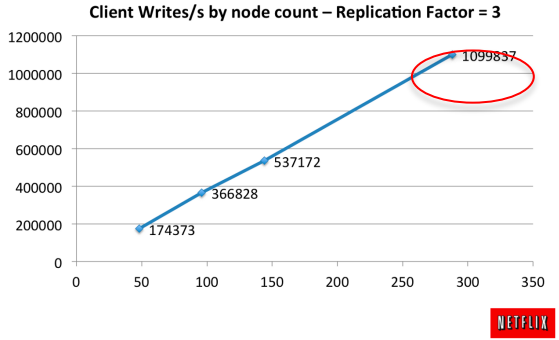
It is located in 9 geographical '**Regions**'.

- Each region is contained within a single country and all of its data and services stay within the designated region.
- Each Region has multiple 'Availability Zones', which are distinct data centers providing AWS services



Distributed Information System – Cassandra

Scale-Up Linearity



Cassandra is a NoSQL distributed database that delivers the high availability, scalability and performance.

It is an excellent example of distributed technology in linear scalability.

- According to Benchmark result from Netflix Technology Team, Cassandra can support over a million writes per second when the node count is 288.

Distributed System in Life: Soccer Team [1]

- Autonomous Individual Players
- Single Coherent Football Team

- Target some Common Objectives
- Players have their Local View(s)
- Global Decisions are Collective
- Communicate through Messages
- Needs Cohesive Synchronization
- Needs some level of Fault Tolerance

[1] reading: http://muratbuffalo.blogspot.com/2020/01/distributed-systems-analogies.html

Distributed Systems

Characteristics and Design Goals

Major Characteristics

Characteristic 1 : Collection of autonomous computing elements

- Modern distributed systems consist of various kinds of nodes.
- Fundamentally, nodes can act independently from each other.
- In practice, nodes are programmed to achieve common goals.

Characteristic 2 : Appears to the users as a single coherent system

- The overall system should appear as a single coherent system.
- The collection of nodes operates as one entity for the end user.
- Ideally, users may not even notice that there are multiple nodes.

Nodes in a Distributed System

Nodes are programmed to achieve common goals by exchanging messages with each other. Nodes react to incoming messages, process them, and in turn, lead to further communication with other nodes through message passing.

- Each node is independent, with an independent notion of time. Thus, we cannot always assume the existence of a global clock for synchronization.
- Managing the membership of nodes within the system is non-trivial, and a distributed system may follow either open-group or a closed-group strategy.
- Nodes are connected through an overlay network, which in principle, should be connected to allow a communication path between any pair of nodes.

Coherent Collective of Nodes

Ideally, the collective of nodes in a system should present a single-system view for the user, where the system behaves according to the expectations of users.

- End-user may not be able to tell exactly on which node or component of the system a process is executing, or if there are processes spawned off from it.
- Allocation of resources, access to nodes, replication of data, or redundancy in execution are part of distribution transparency, a crucial consideration.
- Although partial failures are inherent to any sufficiently complex system, it is extremely hard to hide the effect of such failures in a distributed system. While some applications will run smoothly, some others may fail altogether.

Design Goals

Resource sharing

Distribution transparency

Being open

Being scalable

"Just because it is possible to build distributed systems does not necessarily mean that it is a good idea."

– Distributed Systems (2018)
van Steen and Tanenbaum

Resource sharing

One of the most important goals for distributed systems is to make it easy for the users and their applications to access and share remote resources.

- Ease of collaboration and information exchange
- Collaborative work, meetings, contributions, etc.
- Peer-to-peer file-sharing networks like BitTorrent
- Backup services and data synchronization tools

Design Goals

Resource sharing

Distribution transparency

Being open

Being scalable

"Just because it is possible to build distributed systems does not necessarily mean that it is a good idea."

– Distributed Systems (2018)
van Steen and Tanenbaum

Distribution transparency

Distributed systems aim at making the distribution of processes and resources (objects) transparent (invisible) to the end users and their applications.

Transparency of

- Access : Hides how an object is accessed
- Location : Hides where an object is located
- Migration : Hides if an object is relocated/moved
- Replication : Hides that an object is replicated
- Concurrency : Hides concurrent access/sharing
- Failure : Hides the failure/recovery of an object

Design Goals

Resource sharing
Distribution transparency
Being open
Being scalable

"Just because it is possible to build distributed systems does not necessarily mean that it is a good idea."

– Distributed Systems (2018)
van Steen and Tanenbaum

Being open

An open distributed system offers components that can easily be used by or can easily be integrated into other systems (vice versa in most systems).

- **Interoperability, composability, and extensibility**
  Open systems are built on components adhering to standard rules of syntax and semantics, which facilitates easy configuration and usability.
- **Separation between policy from mechanism**
  Flexible systems are built as a collection of relatively small, easily adaptable components, rather than as a huge monolithic single program.

Design Goals

Resource sharing

Distribution transparency

Being open

Being scalable

"Just because it is possible to build distributed systems does not necessarily mean that it is a good idea."

– Distributed Systems (2018)
van Steen and Tanenbaum

Being scalable

Scalability has become a fundamental goal for modern distributed systems, especially due to the heterogeneity of nodes, resources and components.

Scalability of

- Size : Addressing computational and storage issues with more users, nodes or resources.
- Geography : Addressing communication issues with physically distant nodes and resources.
- Administration : Addressing issues of managing systems spanning independent organizations.

Distributed Systems

Architecture and Communication

System Architecture

Placement of components and deciding on their mutual interactions lead to designing the overall software architecture in case of distributed systems.

Centralized organization

- Simple client-server architecture, with request-reply communication.
- Multitiered architecture, distributing UI, Processing and Data layers.

Decentralized organization

- Structured peer-to-peer systems, with a specific topology for overlay.
- Unstructured peer-to-peer systems, with random graph like overlay.
- Hierarchical peer-to-peer systems, with super-peer overlay network.

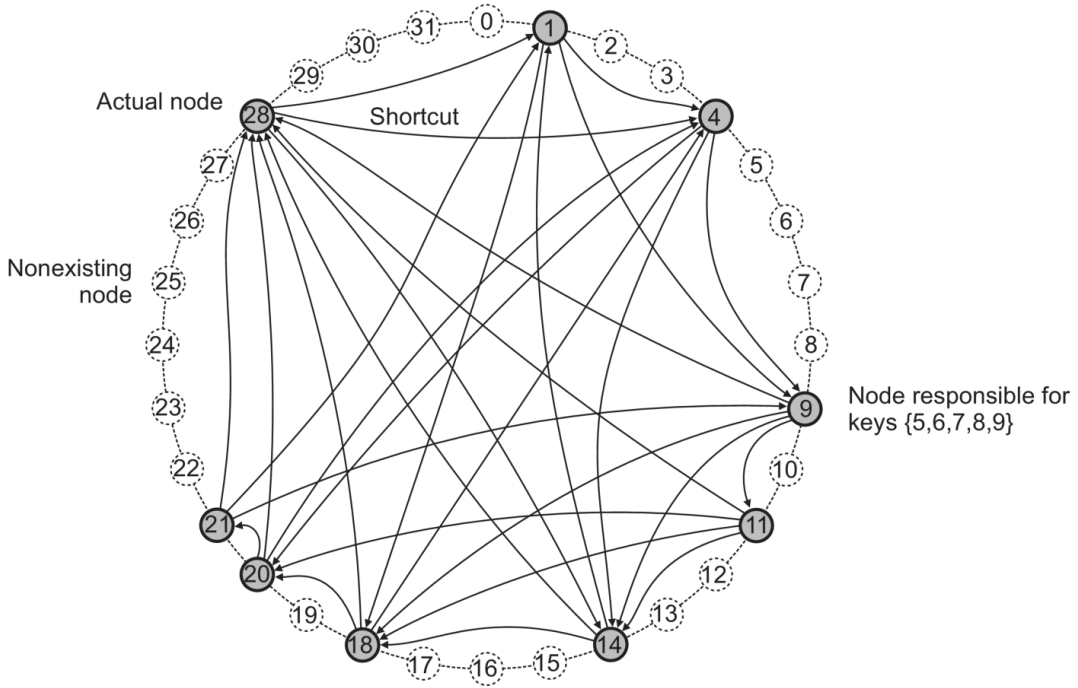
Structured Peer-to-Peer Systems

Specific deterministic topology (e.g. a ring, binary tree or grid).

Each data item is uniquely associated with a key/index.

key(*item*) = *hash*(*value*)

Implemented like a distributed hash table (e.g. the Chord ring).



Unstructured Peer-to-Peer Systems

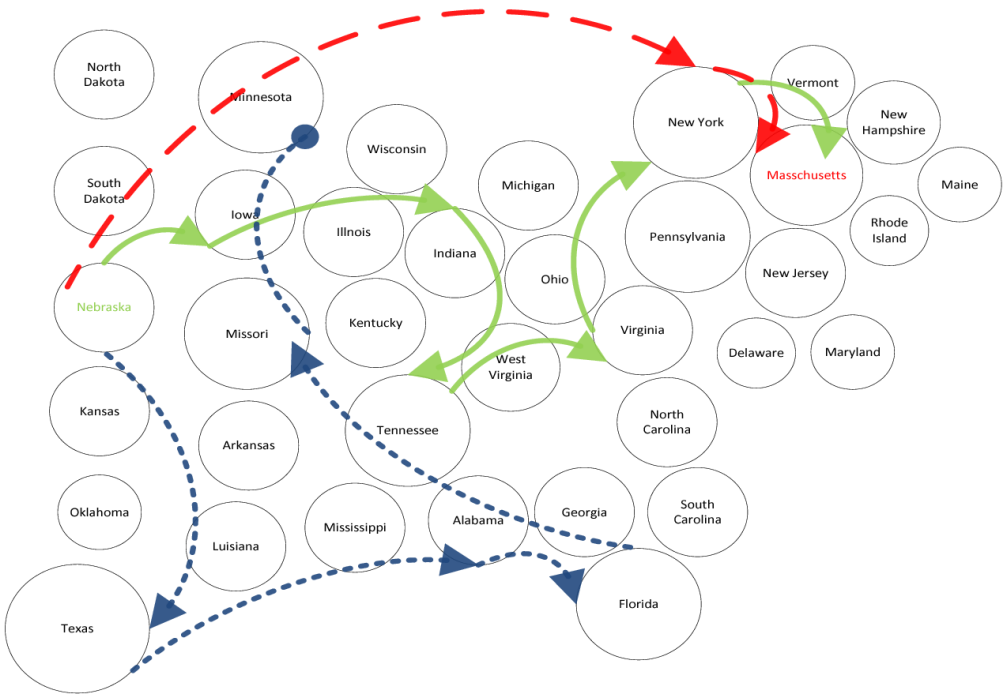
Each node maintains an ad hoc list of neighbors.

(random graph overlay)

Communication extremes:

- Flooding to all Neighbors
- Random Walk on Network

Flexible but Inefficient structure.

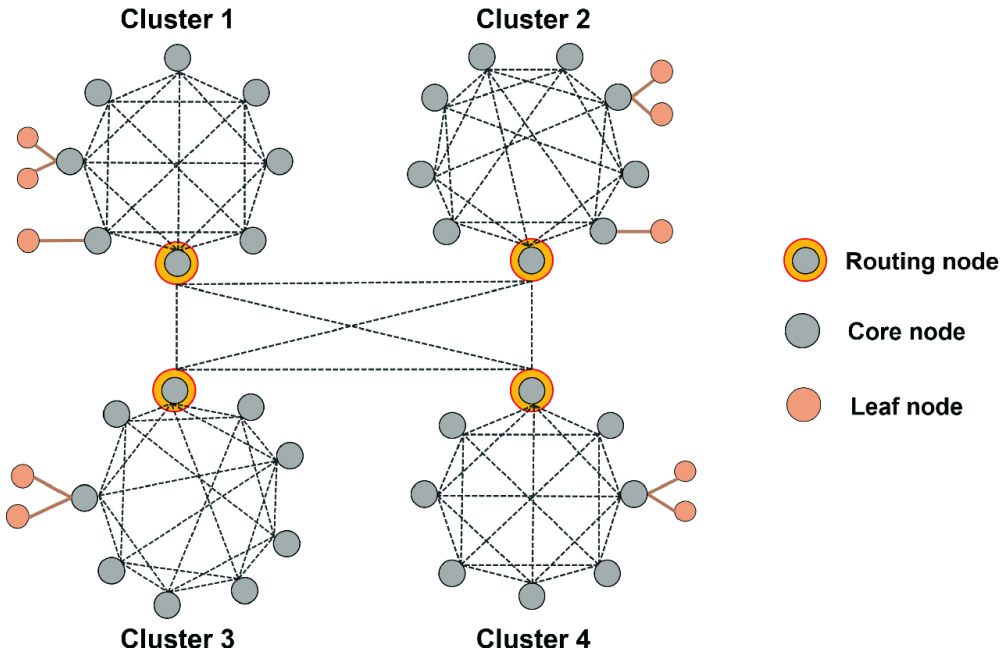


Hierarchical Peer-to-Peer Systems

Overlay network of Super-Peers (e.g. routing nodes or brokers)

Communication is easier, but Super Peer selection becomes a “Leader Election Problem”.

Offers a scalable practical way to model enterprise systems.



P2P Communication

Most P2P networks rely on multicast communication rather than point-to-point. Major issue is setting up communication paths for information dissemination.

Application-level Multicasting : Assumes that nodes are in an overlay network.

Flooding-based Multicasting : Broadcast to all neighbors and check duplicates.

Gossip-based Dissemination : Exchange information with random-picked node.

Gossiping (or a slight variant, Rumor Spreading) is a great way to efficiently and pervasively disseminate information in an Unstructured P2P network.

Analogy : Football/Soccer Team

What is the Distributed Architecture?

Single Coherent Football Team

What is the Communication Model?

Autonomous Individual Players

- Players have their Local View(s)
- Communicate through Messages

Distributed Systems

Consistency and Replication

Need for Replication

Reliability : Data are replicated to increase reliability of a system.

Example : If a file system is replicated, it will still work even if one or more replicas crash. Basic motivation for replication is redundancy.

Distributed systems naturally avoid single-point-of-failure scenarios.

Performance : Replication for performance is crucial for scalability.

Example : If the number of processes accessing a specific data or resources increases, replicas may be used to balance access load.

Poses a trade-off between performance and maintaining consistency.

Data-centric Consistency Models

Potential inconsistencies range across

- deviation in numerical values between replicas,
- deviation in staleness between replicas, and
- deviation with respect to the ordering of update operations on the replicas of the same data or asset.

Continuous Consistency strives to track consistency deviations across replicas. While numerical values and staleness are easy to handle, updates are tricky! Sequential Consistency is crucial for maintaining the global view of a system.

Sequential Consistency

Lamport, 1979 : “The result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its program.”

Process P_1	Process P_2	Process P_3
x ← 1;	y ← 1;	z ← 1;
print(y,z);	print(x,z);	print(x,y);

Valid Execution Sequences

Execution 1	Execution 2	Execution 3	Execution 4
P_1: x ← 1;	P_1: x ← 1;	P_2: y ← 1;	P_2: y ← 1;
P_1: print(y,z);	P_2: y ← 1;	P_3: z ← 1;	P_1: x ← 1;
P_2: y ← 1;	P_2: print(x,z);	P_3: print(x,y);	P_3: z ← 1;
P_2: print(x,z);	P_1: print(y,z);	P_2: print(x,z);	P_2: print(x,z);
P_3: z ← 1;	P_3: z ← 1;	P_1: x ← 1;	P_1: print(y,z);
P_3: print(x,y);	P_3: print(x,y);	P_1: print(y,z);	P_3: print(x,y);
Prints: 001011	*Prints:* 101011	*Prints:* 010111	*Prints:* 111111
Signature: 00 10 11	*Signature:* 10 10 11	*Signature:* 11 01 01	*Signature:* 11 11 11

State Machine Replication [2]

State Machine stores a state of the system at any given time

- It receives a set of inputs, generally in the form of commands.
- The state machine applies these inputs/commands in a sequential order using a transition function to generate an output and an updated state.

Most common Blockchains are examples of State Machine Replication (SMR), where certain Fault Tolerant Consensus schemes are applied for consistency.

> Example : In Bitcoin distributed ledger, the state consists of a set of public keys (addresses) along with the associated Bitcoins recorded in the ledger (to be covered later in the course).

[2] reading: https://decentralizedthoughts.github.io/2019-10-15-consensus-for-state-machine-replication/

Analogy : Football/Soccer Team

- Autonomous Individual Players
- Single Coherent Football Team

Is there a need for Consistency?

Is there a need for Replication?