1 Introduction

1.1 Distributed Systems Definition

A distributed system in its simplest definition is a group of computers working together as to appear as a single computer to the user.

1.2 Why Distributed Systems

- Scaling
  - Vertical: more memory, faster CPU
  - Horizontal: more machines
- Economics
  - Initially scaling vertically is cheaper until max HW
  - Current x86 max: 64 cores
- Location
  - Everything gets faster, latency stays
  - Physically bounded by the speed of light
  - New Protocols can decrease RT
  - Place services closer to user
- Fault tolerance
  - Every hardware will crash eventually

1.3 Scaling

1.3.1 Horizontal

Pros

- Lower cost with massive scale
- Easier to add fault-tolerance
- Higher availability

Cons

- Adaption of software required
- More complex system, more components involved

1.3.2 Vertical

Moore's Law: Nr. of transistors doubles every 2 years.
Nielsen's Law: High-end user's connection speed grows by 50% per year.
Kryder's Law: Disk density doubling every 13 month.
Bandwidth grows slower than computer power

Pros

- Lower cost with small scale
- No adaption of software required
- Less administrative effort

Cons

- HW limits for scaling
- Risk of HW failure causing outage
- More difficult to add fault tolerance

1.4 Distributed Systems Categorization

Tightly Coupled

- Processing Elements have access to a common memory

Loosely Coupled (this lecture)

- Processing Elements have NO access to a common memory

Homogenous System

- All processors are of the same type

Heterogeneous System (this lecture)

- Processors of different types

Small Scale

- WebApp + database

Large Scale (this lecture)

- More than 2 machines

Decentralized

- Distributed in the technical sense, but not owned by one actor

1.4.1 Controlled Distributed Systems

- 1 responsible organization
- Low churn
- Secure environment
- High availability
- Homogenous / Heterogeneous
- Examples: Amazon DynamoDB, Client/Server

Mechanisms that work well:

- Consistent hashing
- Master nodes, central coordinator

Network is under control or client/server

- no NAT issues

Consistency

- Leader election (Zookeeper, Paxos, Raft)

Replication principles

- More replicas: higher availability / reliability / performance / scalability
- Requires maintaining consistency in replicas

Transparency principles apply

1.4.2 Fully Decentralized Systems

- N responsible organizations
- High churn
- Hostile environment
- Unpredictable availability
- Heterogeneous
- Examples: BitTorrent, Blockchain

Mechanisms that work well:

- Consistent hashing (DHTs)
- Flooding/broadcasting - Bitcoin

NAT and direct connectivity huge problem

Consistency

- Weak consistency: DHTs
- Proof of work

Replication / Transparency principles apply

1.4.3 CAP theorem

A distributed data store cannot simultaneously be consistent, available and partition tolerant.

- **Consistency:** Every node has the same consistent state
- **Availability:** Every non-failing node always returns a response
- **Partition Tolerant:** The system continues to be consistent even when network partitions

Examples:

- Network partition: AP or CP
- Blockchain: CP or AP
- Cassandra (Apple): AP, can be configured CP

1.5 Transparency in DS

Distributed system should hide its distributed nature

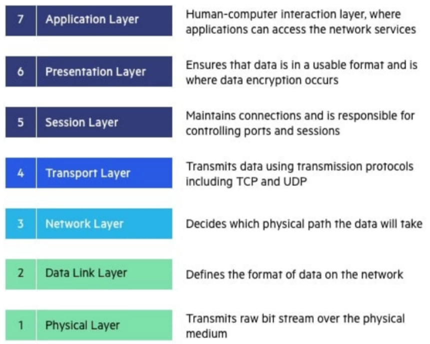
- Location: users should not be aware of the physical location
- Access: users should access resources in a single, uniform way
- Migration, relocation: users should not be aware, that resources have moved
- Replication: Users should not be aware about replicas, it should appear as a single resource
- Concurrency: users should not be aware of other users
- Failure: Users should be aware of recovery mechanisms
- Security: Users should be minimally aware of security mechanisms

1.6 Fallacies of Distributed Computing

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogenous

2 Protocols

2.1 Networking Layers

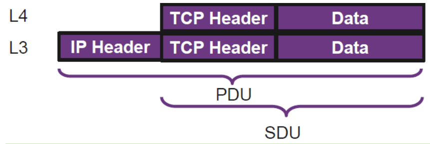


Goal: Interoperability

2.1.1 Layer Abstraction

- Protocols enable an entity to interact with an entity at the same layer in another host
- **Service definitions:** provide functionality to an (N)-layer by an (N-1) layer
- Layer N exchange protocol data units (PDUs) with layer N protocol
- Each PDU contains a header and payload, the service data unit (SDU)

Example PDU of L3:



2.2 Layer 4 - Transport

2.2.1 TCP

- Reliable
- Ordered
- Window - capacity of receiver
- Checksum - 16bit
- TCP overhead: 20 bytes
- Tries to correct errors

Connection establishment

- SYN, SYN-ACK, ACK
- Initiates TCP session: initial sequence number is random

Connection termination

- FIN, ACK + FIN, ACK
- 3-way handshake

Sequences and ACKs

- Identification each byte of data
- Order of the bytes: reconstruction
- Detecting lost data: RTO, DupACK

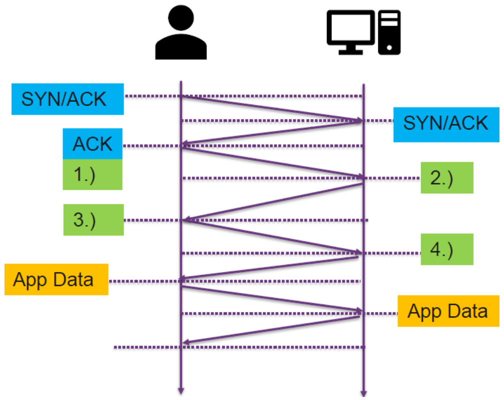
Retransmission timeout

- If no ACK is received after timeout

Flow control

- Sender is not overwhelming a receiver
- Back pressure
- Sliding window
- Congestion control
  - Slow-start
  - Congestion avoidance

2.2.2 TCP + TLS



1. client hello - lists crypto information, TLS version, ciphers/keys
2. server hello - chosen cipher, session ID, random bytes, digital certificate
3. Key exchange using random bytes, now client + server can calculate secret key
4. finished - encrypted message

3RTT until first byte