Table of Contents

Distributed Systems

- **distributed systems**
  - networked system components communicate and coordinate actions through message passing alone
  - collection of independent computers that appears to users as a single, coherent system
- key points
  - *multiple* components in the system
  - *communication* between components
  - *synergy* between components, such that they achieve more than the sum of individual components

Computer Networks vs Distributed Systems

- computer network: independent, interconnected computers that exchange messages based on particular protocols
- distributed system: multiple computers on the network **collaborating** as a system. Spatial separation and communication aspects are hidden from users

Benefits

- **resource sharing**
  - hardware: disks, printers, scanners
  - software: files, databases
  - other: processing power, memory, bandwidth
- **benefits of resource sharing**
  - economy
  - availability
  - reliability
  - scalability

Consequences

- **concurrency:** services provided by DS will be accessed by multiple users simultaneously
  - system design needs to account for this use case, and implement concurrency techniques
- **no global clock:**
  - individual computers have independent clocks
  - limits to the degree of clock synchronisation as all communication is via message passing
  - system design needs to handle absence of global clock, and implement synchronisation techniques
- **independent failures:** some components may fail while others continue to run
  - failure of some components will not be known immediately by others

Wireless networks

Paradigms making heavy use of wireless networks:

- **Mobile/nomadic computing:** users can perform computing tasks on the move, while being part of a DS
- **Ubiquitous computing:** small, cheap embedded computing devices used as part of a DS
- **IoT:** everyday objects are addressable and connected to the Internet.

Challenges

Heterogeneity

- disparate hardware and software. Big challenge
- e.g. different browsers implement many different features
- variation in:
  - networks
  - computer hardware
    - e.g. representation of integer data types: big endian/little endian
  - operating systems: different system calls
  - programming languages
  - implementations by different developers

Approaches

- standard protocols
- agreed upon message formats and data types
- adhering to an API
- using middleware
- portable code (Java)

Middleware

- **middleware:** software layer between distributed application and OS
  - provides programming abstraction
  - masks heterogeneity of underlying resources (hardware, OS, …)
- may only address a subset of heterogeneity issues
- e.g. Java RMI doesn't address programming language heterogeneity

Middleware models

- Distributed file systems
- RPC (procedural languages)
- RMI (OO languages)
- distributed documents
- distributed databases

Mobile Code

- **mobile code** is sent from one computer to another to be run at the destination
  - e.g. Java applets
- code compiled in one OS doesn't run on another architecture, OS
- **Virtual machine** approach: compiler produces code interpreted by the VM, allowing code executable on any hardware
- cross-platform compilation also works

Openness

- **openness:** ability to extend the system by adding hardware/software resources

- e.g. Skype: closed
- e.g. Internet: open protocols has allowed it to scale

Approaches

- publish key interfaces
- ensure all implementations adhere to published standards

Security

- **Confidentiality**: protection against disclosure to unauthorised individuals
- **Integrity:** protection against alteration/corruption
- **Availability:** protection against interference with means of access (e.g. DoS)
  - measure of proportion of time system is available for use
- security against mobile code: executables as attachments

Approaches

- encryption (e.g. RSA)
- authentication (e.g. passwords, public key authentication)
- authorisation (e.g. Access control lists)

Scalability

- **scalability:** ability to handle growth of number of users
- **controlling cost of resources:** ideally scales linearly with number of users
  - always will have some overhead to manage resources
- **controlling performance loss:** dependent on underlying distributed algorithms
  - $O(n^2)$ would not be scalable
  - $O(\log n)$ is scalable
- **preventing resources running out:** e.g. IPv4 addresses; use of particular data types
- **avoiding performance bottlenecks:** resolved with decentralised architectures/algorithms

Failure Handling

- **detection**:
  - some failures can be detected e.g. with checksums
  - others are hard to be certain of, e.g. remote server failure
- **masking**: hide/ameliorate effects of failure
  - e.g. message retransmission after timeout
  - e.g. Zoom may mask errors by interpolating frames
- **tolerating**: sometimes better to tolerate errors, rather than try to handle them
  - e.g. render video with a frame dropped, or report failure to user
- **recovery**: ability to rollback if failure produces corrupted data
- **redundancy**: tolerate failure through redundancy
  - **failover:** multiple servers provide the same service

Concurrency

- multiple clients accessing a shared resource at the same time
- sequential access could handle it but slows down system
- semaphores used by OS to handle concurrency
- e.g. starting up a system also poses concurrency issues: comment that if all of Google was switched off it couldn't be restarted

Transparency

- hiding components of distributed system from user and application programmer
  - system doing something at a lower level that isn't seen by the user
- e.g. on a mobile phone, while moving around, the hardware will change between cells and frequencies without user being aware of it
- sometimes you don't want transparency: some failures need to involve the user e.g. if the Ethernet cable is unplugged
- **network transparency:** combination of access/location transparency. Most important: strongly affects utilisation of distributed resources

  - e.g. email address `username@domain.com`: consists of username and domain name. To send mail you don't need to know physical/network location of mail server, and it is independent of the location of the recipient

- **access transparency**: e.g. distributed file system. API you use to access files locally should be the same as the API you use to access remotely. That way applications don't need to change depending on whether you're accessing locally/remotely.
  - e.g. GUI with folders behaves/looks the same whether files are local or remote
- **location transparency:** access resources without knowledge of physical/network location
- **mobile transparency:** movement of resources/clients in a system without affecting operation of users or programs
  - e.g. moving between cell towers while on a phone call: participants are unaware of change
- Note: **location transparency** is more about server side-changes, while **mobile transparency** is more about client-side changes
- **concurrency transparency:** several processes can operate concurrently on shared resources without interference
- **replication transparency:** multiple resource instances can be used for reliability and performance without knowledge of replicas by users/application programmers
- **failure transparency:** concealment of faults, so that users can complete their tasks despite failures
- **performance transparency:** reconfiguration of system to improve performance as load varies
- **scaling transparency:** allows system to expand in scale without change to system structure or applications

Quality of Service

- main non-functional properties:
  - reliability
  - security
  - performance
  - adaptability
  - availability

Web Services and APIs

- **web services:** allows programs (other than browsers) to be clients for web programs
- data passed as XML/JSON
- **SOAP protocol** allows clients to invoke web services (XML format)
- **REST (REpresentational State Transfer) architecture:**