Lecture 1
CMPU4021 Distributed Systems

Distributed Systems: Introduction

What is a Distributed System (DS)?

- Multiple computers working together on one task
- Computers are connected by a network, and exchange information
- Key motivation for DS is
  - Sharing resources, e.g.
    - data storage,
    - printers, files, databases,
    - programs/applications,
    - multimedia services: camera video/image, frames, audio/phone data

What is a Distributed System (DS)?

- A distributed system is a **collection of autonomous computing elements** that appears to its users as a **single coherent system**.
  - No shared operating system
  - No shared memory
  - No shared clock
  - Transparency of independent failure
- Autonomous computing elements- **nodes**
  - Hardware devices or software processes
- Single coherent system:
  - Users or applications perceive a single system
  - Nodes need to collaborate.

Collection of autonomous nodes

- Independent behaviour
  - Each node is autonomous and will thus have its own notion of time:
    - there is no global clock.
- Collection of nodes
  - How to manage group membership?
  - How to know that you are indeed communicating with an authorized (non)member?

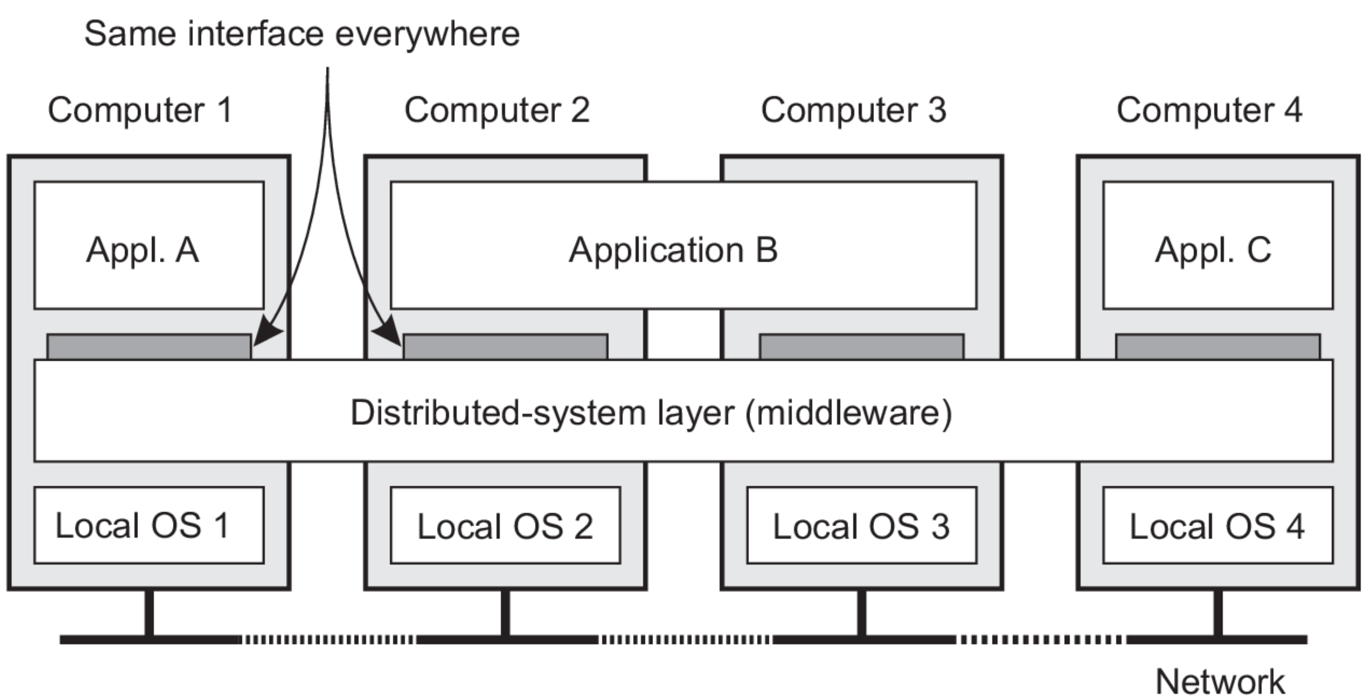
Coherent system

- Essence
  - The collection of nodes as a whole operates the same
    - no matter where, when, and how interaction between a user and the system takes place.
  - Examples
    - An end user cannot tell where a computation is taking place
    - Where data is exactly stored should be irrelevant to an application
    - If or not data has been replicated is completely hidden
- Keyword is
  - distribution transparency
- The drawback: partial failures
  - It is inevitable that at any time only a part of the distributed system fails.
  - Hiding partial failures and their recovery is often very difficult and in general impossible to hide.

Middleware and distributed systems

- Middleware - the OS of distributed systems
  - middleware is the same to a distributed system as what an operating system is to a computer
  - a manager of resources offering its applications to efficiently share and deploy those resources across a network.

- What does it contain?
  - Commonly used components and functions that need not be implemented by applications separately.



DS Characteristics

- Concurrency
  - Collaborative and cooperative problem-solving (interdependencies)
- Independent failures of components
  - Autonomous but interdependent - requires coordination, graceful degradation
- Lack of global clocks
  - Each node is autonomous and will thus have its own notion of time: there is no global clock.
  - Leads to fundamental synchronization and coordination problems.
- Heterogeneity
  - Hardware/software (programs, data) variations in component systems

DS Characteristics I

- Openness
  - Modularity, architecture and standards allow extensibility
- Security
  - Protection (internal and external) against malicious use or attack – integrity
- Scalability
  - Accommodation of increased users and resource demands over time
- Transparency
  - Hide separation of components (hidden by middleware)

Scalability: more performance

- Our computing needs exceed CPU advances
  - Movie rendering
  - Google
    - More than 63,000 search queries per second on average
    - Uses hundreds of thousands of servers to do this
  - Facebook
    - About 100M requests per second with 4B users

- What if we need more performance than a single CPU?
  - Combine CPUs – multiprocessors, or
  - Create distributed systems

Types of distributed systems

- High performance distributed computing systems
  - Started with parallel computing
  - Distributed shared memory systems
  - Cluster computing
    - a group of high-end systems connected through a LAN
  - Grid computing
  - Cloud computing

- Distributed information systems
  - Integrating applications

- Distributed systems for pervasive computing

Distributed information systems

Integrating applications

- Situation
  - Organizations confronted with many networked applications, but achieving interoperability was painful.
- Basic approach
  - A networked application is one that runs on a server making its services available to remote clients.
  - Simple integration: clients combine requests for (different) applications; send that off; collect responses, and present a coherent result to the user.
- Next step
  - Allow direct application-to-application communication, leading to Enterprise Application Integration.

Distributed pervasive systems

- Emerging next-generation of distributed systems in which nodes are small, mobile, and often embedded in a larger system, characterized by the fact that the system naturally blends into the user's environment.

- Supported/spurred by advances in embedded system design
  - device miniaturization and wireless networking.

- Three (overlapping) subtypes
  - Ubiquitous computing systems
    - pervasive and continuously present, i.e., there is a continuous interaction between system and user.
  - Mobile computing systems
    - pervasive, but emphasis is on the fact that devices are inherently mobile.
  - Sensor (and actuator) networks:
    - pervasive, with emphasis on the actual (collaborative) sensing and actuation of the environment.

Examples of DS - The Internet

- Interconnected computers/intranets, which communicate over backbone/medium via messages using the IP (Internet protocol)
  - A *backbone*
    - a network link with a high transmission capacity, employing satellite connections, fibre optic cables and other high-bandwidth circuits.

- Medium:

  - wired: copper circuits, fibre optic

  - wireless: satellite

Examples of DS: Multiplayer Games

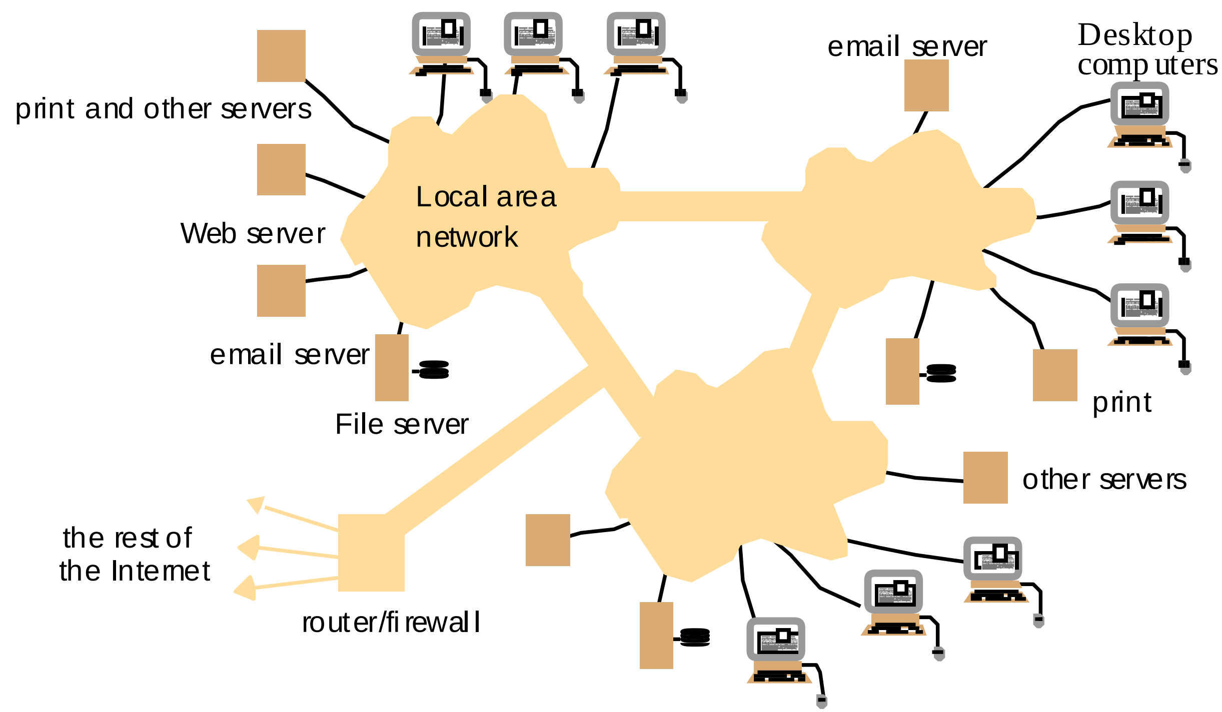
- Different players are doing different things
- Their actions must be consistent
  - Don’t allow one person to be at different locations in views of different people
  - Don’t let two people stand at the same spot
  - If X shoots Y, then everyone must know that Y is dead
- Made difficult by the fact that players are on different computers
- Sometimes network may be slow
- Sometimes messages can be lost

Examples of DS: Stock markets

- Everyone wants information quickly and to buy/sell without delay
- Updates must be sent to many clients fast
- Transactions must be executed in right order
- Specialized networks worth millions are installed to reduce latency

Examples of DS - Intranets

- Locally owned interconnected computers – *may be* a component of a DS, e.g., the Internet – if so, the intranet is connected via a router.
- Typically composed of LANs, with firewall (if connected to the Internet) for filtering incoming/outgoing transmissions



Examples of DS: Web Search

- Google – a sophisticated distributed system infrastructure
  - A very large numbers of networked computers located at data centres all around the world;
  - A distributed file system designed to support very large files and heavily optimised for the style of usage required by search and other Google applications
  - An associated structured distributed storage system that offers fast access to very large datasets;
  - A lock service that offers distributed system functions such as distributed locking and agreement.

Examples of DS: Massively multiplayer online games (MMOGs)

- A number of solutions
  - EVE Online (CPP Games)
    - Client-server architecture
    - A single copy of the state of the world is maintained
  - Other MMOGs (e.g. EverQuest) adopt
    - More distributed architecture with potentially large number of servers
  - Researchers are now looking at more radical architecture
    - Based on peer-to-peer technology

DS Challenges

- Fundamental issue
  - Different nodes have different knowledge.
  - One node does not know the status of other nodes in the network

- If each node knew exactly the status at all other nodes in the network, computing would be easy:
  - Impossible, theoretically and practically

Theoretical issue: Knowledge cannot be perfectly up to date

- Information transmission speed
  - hardware and software limitations of the nodes & network
- New things can happen while information is traveling from node A to node B
- B can never be perfectly up to date about the status of A

Practical Challenges

- Communication is costly
  - It is not practical to transmit everything from A to B all the time
- There are many nodes
  - Transmitting updates to all nodes and receiving updates from all nodes are even more impractical

DS Challenges – Heterogeneity I

- Heterogeneity (variety and difference) applies to:
  - Networks
    - differences are masked by the fact that all of the computers use the Internet protocols to communicate.
  - Hardware
    - data types, such as integers, may be represented in different ways on different sorts of hardware (byte ordering: big-endian, little-endian)
  - Operating systems
    - do not provide the same application API to the Internet protocols.
  - Programming languages
    - used different representations for characters and data structures, such as arrays and records.
  - Developers
    - representation of primitive data items and data structures needs to be agreed upon (standards)

DS Challenges – Heterogeneity I

- Middleware
  - Software layer that provides a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware, operating systems and programming languages
  - All middleware deals with the differences in operating systems and hardware.

- Mobile code
  - The code that can be sent from one computer to another and run at the destination.
    - Machine code suitable for running on one type of computer hardware is not suitable for running on another
  - Virtual machines approach
    - provides a way of making code executable on any hardware: the compiler for a particular language generates code for a virtual machine instead of a particular hardware order code.

DS Challenges - Openness

- The characteristic that determines whether the system can be extended and re-implemented in various ways.
- Can it be extended with new content/services without disruption to the underlying system?
- Key interfaces/standards are published
- Standards
  - Request For Comments (RFC)s
  - IETF
  - W3C
- Everything conforms to a standard!

DS Challenges – Openness I

- *Open systems*
  - key interfaces are published.

- *Open distributed systems* are based on the provision of a uniform communication mechanism and published interfaces for access to shared resources.
  - They can be constructed from heterogeneous hardware and software
    - possibly from different vendors

DS Challenges - Security

- Three main issues
  - Confidentially
    - protection against disclosure to unauthorized individuals
  - Integrity
    - protection against alteration or corruption
  - Availability
    - protection against interference with the means to access the resources

DS Challenges – Security I

- Denial of Service
- Viruses/Worms
- Password Crackers
- Packet sniffers
- Trojan horses
- Spoofing
- Mobile Code

DS Challenges - Scalability

- A system is scalable if it will remain effective if there is a significant increase in the number of resources and the number of users

- Control cost of physical resources
  - *For a system with n users to be scalable, the* quantity of physical resources required to support them should be at most O(*n*) – that is, proportional to n.
    - E.g., if a single file server can support 20 users, then two such servers should be able to support 40 users.

DS Challenges – Scalability I

- Avoid performance bottle neck
  - Algorithms should be decentralised to avoid having performance bottlenecks.

    Predecessor of the Domain Name System (DNS) kept the name table in a single master file that could be downloaded to any computers that needed it - fine with a few hundred computers.

    The DNS removed the bottleneck by partitioning the name table between servers located throughout the Internet and administered locally.

- Scalability techniques:

  - Replicated data, caching, multiple servers etc.

DS Challenges - Failure Handling

- Failures in distributed systems are mostly partial failures which can make failure handling more difficult

- Detecting failures
  - Checksums

- Masking failures
  - retransmission
  - duplicate files

DS Challenges - Failure Handling I

- Tolerating failures
  - Web pages (informing users about failure)

- Recovery

  - permanent data ‘rolled back’

- Redundancy (use of redundant components)
  - Duplication in routes, hardware,
  - DNS – every name table replicated in at least two different servers,
  - Databases – replicated in several servers

- Availability – measure of the proportion of time a system is available for use.

- DS provide a high degree of availability regarding hardware faults.

DS Challenges - Concurrency

- Resources can be shared by clients in a distributed system
  - several clients may access a shared resource at the same time
- Not acceptable that each request be processed in turn
  - must be able to process requests concurrently
- Data consistency
  - For each ‘object’ that represents a shared resource, its operations must be synchronised in such a way that its data remains consistent

Transparencies

- Hide distribution from users and from software

- The concealment from the user and application programmer of the separation of components in a distributed system:
  - Access transparency
  - Location transparency
  - Concurrency transparency
  - Replication transparency
  - Failure transparency
  - Mobility transparency
  - Performance transparency
  - Scaling transparency

DS Challenges – Transparency I

- Access transparency
  - Enables local and remote resources to be accessed using identical operations.
  - A GUI with folders, which is the same whether the files are local or remote.

- Location transparency:
  - enables resources to be accessed without knowledge of their location
  - Web resource names or URLs – because the part of the URL that identifies a web server domain name refers to a computer name in a domain, rather than to an Internet address.

- Concurrency transparency
  - Enables several processes to operate concurrently using shared resources without interference between them.

DS Challenges – Transparency II

- *Replication transparency*
  - enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.
- *Failure transparency*:
  - enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.
    - Email is eventually delivered, even when servers or communication links fail.

DS Challenges – Transparency III

- *Mobility transparency*
  - allows the movement of resources and clients within a system without affecting the operation of users or programs.
- *Performance transparency*
  - allows the system to be reconfigured to improve performance as loads vary.
- *Scaling transparency*
  - allows the system and applications to expand in scale without change to the system structure or the application algorithms.

Distributed Systems - Issues I

- Load balancing
  - directing clients to servers with light loads
- Transparent failover
  - server crashes, how fast can clients be rerouted to other servers without interruption
- Backend integration
  - legacy systems

Distributed Systems - Issues II

- Transactions
  - simultaneous updates in database when it crashes
- Clustering
  - server contains state when it crashes, is that state replicated across all servers?
- Dynamic redeployment
  - how do you perform updates with the system is running?

Distributed Systems - More Issues III

- Clean shutdown
  - can you shutdown a server without disruption to a client?
- Logging and Auditing
  - if something goes wrong are their logs available for debugging purposes?
- System Management
  - if there is a catastrophic error who is notified and how?

Distributed Systems - Issues IV

- Threading – how do we implement threading to facilitate concurrency
- Object lifecycle – when do objects in the system need to be created/destroyed
- Resource pooling – if a client is not using a resource, it should be returned to the pool of resources
- Caching – if particular information is being requested constantly, should and how can we cache it?

Summary

- A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system.
- Resource sharing is the main motivating factor for constructing distributed systems.
- Make it easier to integrate different applications running on different computers
- If designed properly, they can scale well
- Often comes at the cost of more complex software, degradation of performance and weaker security
- Middleware is a key facility for the building of distributed systems

References

- Chapter 1: Coulouris, Dollimore and Kindberg, Distributed Systems: Concepts and Design, 5/E

- Chapter 1: Maarten van Steen, Andrew S. TanenbaumDistributed Systems, 3rd edition (2017)