What is a distributed system?

- **Definition [Coulouris & Emmerich]**
  - A distributed system consists of hardware and software components located in a network of computers that communicate and coordinate their actions only by passing messages.

- **Definition [Lamport]**
  - A distributed system is a system that prevents you from doing any work when a computer you have never heard about, fails.
Types of distributed system

- Distributed Computing Systems
  - Used for high performance computing tasks
  - Cluster computing systems
  - Grid computing systems
- Distributed Information Systems
  - Systems mainly for management and integration of business functions
  - Transaction processing systems
  - Enterprise Application Integration
- Distributed Pervasive Systems
  - Mobile and embedded systems
  - Home systems
  - Sensor networks

A distributed system organized as middleware

- Layer of software offering a single-system view
- Offers portability and interoperability
- Simplifies development of distributed applications and services

DISTRIBUTION MIDDEWARE

Platform Independent API

- Distributed applications and services
  - transaction oriented (ODTP XA)
  - message oriented (IBM MQSeries)
  - remote procedure call (X/Open DCE)
  - object-based (CORBA, COM, Java)

Platform Dependent API

Local OS 1  Local OS 2  ...  Local OS n
Implications of distributed systems

- Independent failure of components
  - “partial failure” & incomplete information
- Unreliable communication
  - Loss of connection and messages. Message bit errors
- Unsecure communication
  - Possibility of unauthorised recording and modification of messages
- Expensive communication
  - Communication between computers usually has less bandwidth, longer latency, and costs more, than between independent processes on the same computer
- Concurrency
  - Components execute in concurrent processes that read and update shared resources. Requires coordination
- No global clock
  - Makes coordination difficult

Requirements leading to distributed systems

- Resource sharing
  - The possibility of using available resources anywhere
- Openness
  - An open system can be extended and improved incrementally
- Scalability
  - Serve more users, provide shorter response times
- Fault tolerance
  - Maintain availability even when individual components fail
- Heterogeneity
  - Network and hardware, operating system, programming languages, implementations by different developers
Resource sharing

- The opportunity to use available hardware, software or data anywhere in the system
- Resource managers control access, offer a scheme for naming, and controls concurrency
- A resource manager is a software module that manages a resource of a particular type
- A resource sharing model describes how
  - resources are made available
  - resources can be used
  - service provider and user interact with each other

Models for resource sharing

- Client-server resource model
  - Server processes act as resource managers, and offer services (collection of procedures)
  - Client processes send requests to servers
- Object-based resource model
  - Any entity in a process is modeled as an object with a message based interface that provides access to its operations
  - Any shared resource is modeled as an object
Openness

- An open DS can be extended and improved incrementally
- Requires a uniform IPC mechanism and publication of component interfaces (e.g., subject to standardisation)
  - IETF RFC: Protocol specifications (www.ietf.org)
  - OMG: interface specifications etc. (www.omg.org)
- New components can be integrated with existing components

Concurrency

- Components in DS execute in concurrent processes
- Components access and update shared resources (e.g., variables, data bases, device drivers)
- Integrity of the system may be violated if concurrent updates are not coordinated
- Preservation of integrity requires concurrency control where concurrent access to the same resource is synchronized
**Scalability**

- A system is scalable if it remains effective when there is a significant increase in the amount of resources and number of users
  - Internet: no of users and services has grown enormously
- Scalability denotes the ability of a system to handle an increasing future load
- Requirements of scalability often leads to a distributed system architecture (several computers)

---

**Scalability : challenges**

- Controlling the costs of physical resources
  - A system with \( n \) users is resource-scalable if the amount of resources required to support them is at most \( O(n) \)
- Controlling the performance loss (when the amount of data increases)
  - A system is performance scalable if the time it takes to access hierarchically ordered data is at most \( O(\log n) \) where \( n \) is the amount of data
- Preventing software resources running out:
  - Dimension data structures o.a. such that the system can meet future requirements (difficult – cf. IP addresses)
- Avoiding performance bottlenecks
  - require decentralized algorithms (partitioning, caching and replication)
Failure handling

- Hardware, software and network fail!!
- DS must maintain availability even in cases where hardware/software/network have low reliability
- Failures in distributed systems are partial
  - makes error handling particularly difficult
- Many techniques for handling failures
  - Detecting failures (checksum o.a)
  - Masking failures (retransmission in protocols, replication ...)
  - Tolerating failures (as in web-browsers)
  - Recovery from failures
  - Redundancy (replicate servers in failure-independent ways)

Heterogeneity

- Variations and differences that must be handled
  - network
    - The Internett-protocol is implemented over many different networks
  - Hardware
    - difference in representation of data types on different processors
  - operating system
    - API to the same protocol and services varies
  - programming languages
    - different representation of character set and data structures
  - implementation by different developers
    - ensure that different programs can communicate
      - requires agreement on a number of things (cf. standards)
**Transparency**

- Transparency hides the consequences of distribution
- Transparency has different dimensions
- These represents different properties a distributed system might have (metric to assess the design of a system)

**Access transparency**

- Enables local and remote resources/components to be accessed using identical operations
- Example: File system operations in NFS
- Example: Navigation in www
- Example: SQL-queries in distributed data bases
- Components that do not have transparent access can not easily be moved to another computer
Location transparency

- Enables local and remote resources/components to be accessed without knowledge of their location
- Example: File system operations in NFS
- Example: Web pages (URLs) in www
- Example: Tables in distributed databases

Other transparency dimensions (Coulouris)

- Concurrency transparency
- Replication transparency
- Failure transparency
- Mobility transparency
- Performance transparency
- Scaling transparency
Summary

- Distributed systems:
  - Hardware and software-components located in a network of computers that communicates and coordinates their actions exclusively by sending messages
  - Consequences of distributed systems
    - Independent failure of components
    - Unsecure communication
    - No global clock
  - Requirements like resource sharing, openness, scalability, fault tolerance and heterogeneity can be satisfied by distributed systems
- Distributed systems organized as middleware
  - Harvest potential advantages of distributed systems without having to pay for all their challenges and problems (transparency)
Many has experience with design of local objects that are all located in the execution environment of a OO programming language.

Design of distributed objects is different!
Design of distributed objects

Local vs distributed objects

- References
- Activation/deactivation
- Migration/mobility
- Persistence
- Latency for method calls
- Concurrency
- Communication
- Security
- Many pit falls!!
Object references

- References to objects in OOPS are usually pointers to memory cells
- References to distributed objects are more complex
  - location information
  - security information
  - references to object type
- References to distributed objects are larger (e.g., 350 byte in Orbix)

Activation/deactivation

- Objects in OOPS reside in main memory during their whole life time
- This does not always suit distributed objects
  - no of objects
  - objects can be used over a long period of time
  - some servers must be shut down from time to time without stopping the applications
- Distributed object implementations are
  - read into main memory (activation)
  - removed from main memory (deactivation)
**Activation/deactivation**

- **Issues:**
  - repository for implementations
  - association between objects and processes
  - explicit vs implicit activation
  - when to deactivate objects?
  - how to handle concurrent calls

- **Who decides?**
  - Designer?
  - Programmer?
  - Administrator?

---

**Persistence**

- Stateless vs stateful objects
- Stateful objects must store its state in a persistent repository between
  - object-deactivation and
  - object-activation
- Can be achieved by
  - making an external representation for file system
  - map to relational database
  - object database
- Decided at object design time
Object life cycle

- Objects in OOPS exist in a single virtual machine
- Distributed objects can be created on different computers
- Distributed objects can be copied or moved from one computer to another
- Removal of objects by “garbage collection” is difficult in a distributed environment
  - Java RMI: “reference counting”
  - Jini: “leases”
- Life cycle must be considered at design time of distributed objects

Latency of method calls

- To execute a local method call requires a few hundred nanoseconds
- A remote method call requires between 0.1 and 10 milliseconds, or more
- ⇒ interfaces of distributed objects must be constructed such that
  - methods perform larger processing tasks
  - highly frequent method calls are not required
**Parallelism**

- Execution of objects in OOPS
  - sequential
  - concurrent with multiple threads
- Distributed objects execute in parallel
- Can be used to accelerate computations

**Communication**

- Method calls in OOPS are synchronous
- Alternatives for distributed objects:
  - synchronous requests
  - oneway requests
  - deferred synchronous requests
  - asynchronous requests
- Who decides for each call?
  - designer of service?
  - designer of client?
- How to document?
Security

- Security in OO applications can be handled at session level
- Objects in OOPS do not have to be written in a particular way
- For distributed objects
  - Who issues the method call?
  - How do we know that the client is the one he claims to be?
  - How can we decide whether to grant the client the right to use the service?
  - How can we prove that we have delivered the service to enable later billing for the use of the service?

Summary

- Design of distributed objects is different from design of programs where all object are located in the same process
- Many pit falls!!