TDDD36

Secure Mobile Systems

Systems Software

Lecture 4: Distributed Systems Models

Simin Nadjm-Tehrani

Real-time Systems Laboratory

Department of Computer and Information Science
Linköping University

In lecture 1...

- We looked at Distributed systems as one possible realisation of Concurrent processes
- This time we look at some basic concepts for modelling, and design principles for distributed systems

Applications

- Banking systems
- On-line access & electronic services
- Peer-to-Peer networks
- Distributed control
  - Cars, Airplanes
- Sensor and ad hoc networks
  - Buildings, Environment
- M2M communication

Common in all these?

Distributed model of computing:

- Multiple processes
- Disjoint address spaces
- Inter-process communication
- Collective goal

Distributed Systems

A distributed system is one in which the failure of a computer you did not even know existed can render your own computer unusable.

Leslie Lamport

Networking vs. Dist. systems

- "Networking" treats the internal mechanisms for inter-process communication:
  - Routing
  - Error control (reliable transmission)
  - Flow control (low level treatment of overloads)

As if it were one computer running a service for a user!

- "Distributed systems" treats the *application view* of the architecture for communication and cooperation

This lecture

- Basic aspects affecting design
- Distributed systems architectures and models

Overview

Why is it hard to get it right?

- Variations in workload, connectivity, mobility, requirements
- Heterogeneity in systems environment, hardware, operating systems, and networks
- Consequences of timing and failure issues
- Security threats, and distributed attacks

Architectural models

- Placement of processes and data across a network of computers
- Patterns of communication to achieve functional and extra(non)-functional properties

What are these?

- Challenges: Scalability, interoperability

Common application models

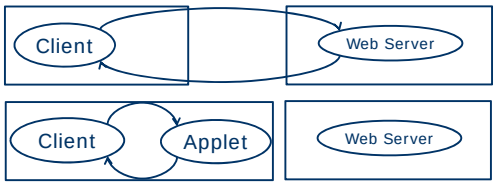
- Client-Server systems
  - Some nodes act as provider of a service and some nodes as requester of a service
  - Multi-tier extensions allow some servers to act as clients of another server in their turn
- Peer-to-Peer systems
  - All nodes can be both a provider and a requester

Variations to basic models

- Multi-tier servers
  - Web server as a client of a local file server
  - DNS servicing other Internet servers
- Multiple-server clusters
  - Service processing divided among replicated on-line store servers
  - Load balancing among IP-telephony servers
  - The cloud
- Proxy servers
  - Acting as front-end or as cache

Mobile code

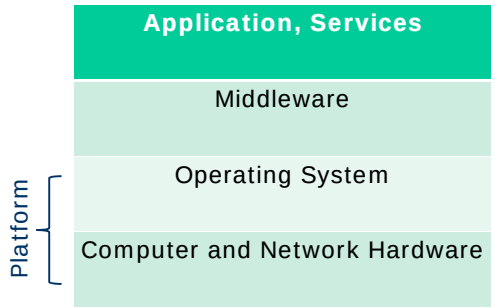
- Local interaction after downloading code from server



Architectural models

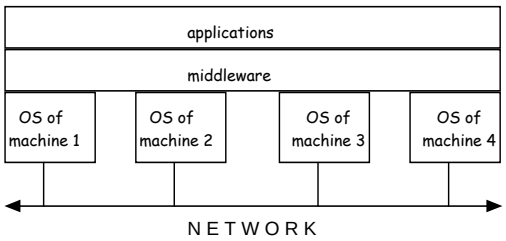
- Placement of processes and data across a network of computers
- Patterns of communication to achieve functional and extra(non)-functional properties
- Challenges: Scalability, interoperability

Interoperability



Programmer's interface

- Middleware masks heterogeneity



The role of middleware

- Providing services that are common for a range of applications
- Least level of service provided
  - Interoperability
  - Location and communication transparency
- But: Heavy middleware diminishes scalability

Design requirements

- Sharing resources
  - Data: consistency & concurrent updates
- Performance
  - **Response time**: service time, middleware services, operating system overheads, communication time
  - **Throughput:** measure of work done, function of load and processing/transfer rate
- Quality of service
  - Adapting performance and dependability

Distributed system models

- What is a model good for?
- What do we know about a design?
  - Assumptions
  - Outcomes: what we can guarantee, and what we cannot guarantee
- Interaction/failure/security model

Two interaction models

- **Asynchronous:** No relation between computation rate at different nodes, No bound on message exchange delay, Clock drift rates are arbitrary
- **Synchronous:** Bounded message exchange delay, Related processing rates at different nodes, Clock drift rates bounded

Implications

- Synchronous:
  - Local clocks can be used to implement timeouts
  - Lack of response from another node can be interpreted as detection of failure

- Asynchronous:
  - In the absence of global (synchronised) time one cannot relate clocks at different nodes

Fundamental models

- What is a model good for?
- What do we know about a design?
  - Assumptions
  - Outcomes: what we can guarantee, and what we cannot guarantee
- Interaction/failure/security model

Failure models

- We will look into more detail into failure and related notions in next lecture
- For now…
- Distributed systems can fail in nodes or channels
- Node failures:
  - Crash
  - Omission
  - timing
  - Byzantine (arbitrary)

Channel failures?

Fundamental models

- What is a model good for?
- What do we know about a design?
  - Assumptions
  - Outcomes: what we can guarantee, and what we cannot guarantee
- Interaction/failure/security model

Security model

- Security in a distributed system can be achieved by securing
  - the nodes/processes
  - the channels used for interaction
  - the encapsulated objects against unauthorised use

Threat model

- Identifying the adversary model helps to identify
  - Threats to channels
  - Threats to processes and potential for denial of service
- This helps to identify appropriate counter measures
  - Secure channels, access control and monitoring activities