Distributed Systems

- Lecturer: Therese Berg `therese.berg@it.uu.se`.
- Recommended text book:

  "*Distributed Systems Concepts and Design*", Coulouris, Dollimore and Kindberg. Addison and Wesley.

- Course webpage to follow.
  http://www.it.uu.se/edu/course/homepage/datsyst2/p2ht06

The slides are put on the webpage.

Course Plan

- Introduction.
- Remote Objects and Remote Procedure calls.
- Replication
- Coordination and Agreement
- Transactions
- Distributed Transactions (if time)
- Guest lecturer - Ambient Networks, mobil communication

What is a distributed System?

> "A distributed system is one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages."

Motivation: share resources.

Consequences of distributed systems

- Concurrency.
- No global Clock.
- Independent Failures.

Concurrency

- Different Computers on a network. Each computer can be doing work at the same time.
- What happens if two computers want to access a resource at the same time?
- Network delays are not constant make synchronisation difficult.

No Global Clock

- When programs need to cooperate they coordinate their actions by exchanging messages.
- Close coordination often depends on a shared idea of the time at which events occur. Ex: *make* command on Unix systems.
- There are limits to the accuracy with which components in a network can synchronise their clocks.
- Direct consequence of the fact that messages are sent through a network and network delays are not constant (or even bounded).

Independent failures

- All computer systems can fail it is good design to build robustness in.
- Distributed systems fail in new ways.
- Network faults might result in components being isolated but not stopping. What happens if you are waiting for an acknowledge message and you never get it? How do you know if the remote system got your message?

Examples of Distributed Systems

- The Internet.
- Intranets. (Internal Internet inside a company)
- Grid Computing.
- P2P network (Peer to Peer).

Resource Sharing

- What is a resource?
  - Hardware. Shared printer. Shared processor
  - Pieces of running software, distributed objects, remote procedures.
  - Data.

Challenges

- Heterogeneity. (Everybody is different).
- Security
- Scalability
- Failure Handling
- Concurrency
- Transparency

Heterogeneity

- Different networks
- Different hardware
- Different operating systems
- Different programing languages
- Different implementations.

How do we solve all this?

Middleware

- A software layer that provides programming abstractions as well as masking the heterogeneity of the underlying system.
- Examples include:
  - CORBA
  - Java RMI, Jini
- How do you deal with mobile code in a heterogeneous environment?
  - One possible solution is to use a virtual machine, Java VM or .Net.

Security

- On its own network traffic is not secure. Any message could in principle be seen by anybody. Packet sniffers.
- Encryption can solve some problems.

More problems include:

- Denial of service attacks.
- How do you make mobile code secure.

Scalability

- Short story, things get bigger all the time. More users, more computers, more data, more ...
- Physical resources.
- Software resources.
- Avoid bottlenecks in performance.
- Design problem.

Failure Handling

- *Detecting Failures*: Some failures can be detected. But in a distributed system it becomes hard.
- *Failure Masking*: Retransmission, backups (data and servers).
- *Tolerating failures*: Klients and users tolerate faults.
- *Recovery*: Server crash, data it updates in inconsistent state.
- *Redundancy*: Use redundant komponents.

Concurrency

- Managing shared resources.
- For an object to be safe in a concurrent environment its operations must be synchronised in such a way that data remains consistent (bank account example).

Transparency

- Access transparency enables local and remote resources to be accessed using identical operations.
- Location transparency. Allows access without knowing object locations.
- Concurrency transparency
- Replication transparency: enables multiple instances to be active to increase reliability and performance.
- Failure transparency
- Mobility transparency.

Architectural Models

- An Architectural model of a distributed system is concerned with the placements of its parts and the relationship between them.
- We are going to look the most popular design for a distributed system. The client server model.

Insert figure 2.2. from the book here.

Client Server Model

- Process acts as a client and sends requests to a server.
- Examples:
  - Webserver. Client (Web browser) sends a request for a webpage the webserver then returns the requested server.
  - A SQL server, client processes send request for data or requests to modify data.
- Servers can become clients. For example the webserver might simply be a web proxy and pass requests on to other servers.

Variations of the client-server model

- Mobile code and Mobile Agents
- Thin clients

Mobile Code and Mobile Agents

.

- Mobile Code.
  - A client request results in downloading of applet code and then the client interacts with the applet.
- Mobile Agents
  - A mobile agent is a running program (code and data) the travels from one computer to another in a network performing some task.

Thin Clients

- A thin client refers to software/hardware the supports a window-based user interface local to the user while executing application programs on a remote computer.
- Examples include:
  - Sun Rays
  - VNC
  - Microsoft remote desktop.

Summary and Conclusion

- Distributed Systems are everywhere.
- Distributed systems have their own design problems and issues.
- Middleware supplies abstractions to allow distributed systems to be designed. Focus of this course: What abstractions are necessary to a distributed system.
- Client-server architecture is a common way of designing distributed systems.