MODELS OF DISTRIBUTED SYSTEMS

1. Architectural Models

2. Interaction Models

3. Fault Models

Basic Elements

- **Resources** in a distributed system are shared between users. They are normally encapsulated within one of the computers and can be accessed from other computers by communication.

- Each resource is managed by a program, the resource manager; it offers a communication interface enabling the resource to be accessed by its users.

- Resource managers can be in general modelled as processes. If the system is designed according to an object-oriented methodology, resources are encapsulated in objects.

Basic Elements

- **Resources** in a distributed system are shared between users. They are normally encapsulated within one of the computers and can be accessed from other computers by communication.

- Each resource is managed by a program, the resource manager; it offers a communication interface enabling the resource to be accessed by its users.

- Resource managers can be in general modelled as processes. If the system is designed according to an object-oriented methodology, resources are encapsulated in objects.

Architectural Models

How are responsibilities distributed between system components and how are these components placed?

- **Client-server model**
- **Peer-to-peer**

Variations of the above two:
- Proxy server
- Mobile code
- Mobile agents
- Network computers
- Thin clients
- Mobile devices

Client - Server

- The system is structured as a set of processes, called servers, that offer services to the users, called clients.

- The client-server model is usually based on a simple request/reply protocol, implemented with send/receive primitives or using remote procedure calls (RPC) or remote method invocation (RMI):
  - the client sends a request (invocation) message to the server asking for some service;
  - the server does the work and returns a result (e.g., the data requested) or an error code if the work could not be performed.
Client - Server (cont’d)

- A server can itself request services from other servers; thus, in this new relation, the server itself acts like a client.

Peer-to-Peer

- All processes (objects) play similar role.
  - Processes (objects) interact without particular distinction between clients and servers.
  - The pattern of communication depends on the particular application.
  - A large number of data objects are shared; any individual computer holds only a small part of the application database.
  - Processing and communication loads for access to objects are distributed across many computers and access links.
  - This is the most general and flexible model.

Peer-to-Peer (cont’d)

- Some problems with client-server:
  - Centralisation of service ⇒ poor scaling
    - Limitations:
      - capacity of server
      - bandwidth of network connecting the server

- Peer-to-Peer tries to solve some of the above
  - It distributes shared resources widely
    \[
    \text{share computing and communication loads.}
    \]

- Problems with peer-to-peer:
  - High complexity due to
    - cleverly place individual objects
    - retrieve the objects
    - maintain potentially large number of replicas.

Variations of the Basic Models

- Client-server and peer-to-peer can be considered as basic models.
  - Several variations have been proposed, with considering factors such as:
    - multiple servers and caches
    - mobile code and mobile agents
    - low-cost computers at the users’ side
    - mobile devices
**Proxy Server**

☞ A proxy server provides copies (replications) of resources which are managed by other servers.

- Proxy servers are typically used as caches for web resources. They maintain a cache of recently visited web pages or other resources. When a request is issued by a client, the proxy server is first checked, if the requested object (information item) is available there.
- Proxy servers can be located at each client, or can be shared by several clients.
- The purpose is to increase performance and availability, by avoiding frequent accesses to remote servers.

**Mobile Code**

☞ Mobile code: code that is sent from one computer to another and run at the destination.

**Advantage:** remote invocations are replaced by local ones.

Typical example: Java applets.

**Step 1:** load applet

**Step 2:** interact with applet

**Mobile Agents**

☞ Mobile agent: a running program that travels from one computer to another carrying out a task on someone's behalf.

- A mobile agent is a complete program, code + data, that can work (relatively) independently.
- The mobile agent can invoke local resources/data.

**Typical tasks:**

- Collect information
- Install/maintain software on computers
- Compare prices from various vendors by visiting their sites.

**Attention:** potential security risk (like mobile code)!

**Network Computers**

☞ Network computers do not store locally operating system or application code. All code is loaded from the servers and run locally on the network computer.

**Advantages:**

- The network computer can be simpler, with limited capacity; it does not need even a local hard disk (if there exists one it is used to cache data or code).
- Users can log in from any computer.
- No user effort for software management/administration.
**Thin Clients**

☞ The thin client is a further step, beyond the network computer:
- Thin clients do not download code (operating system or application) from the server to run it locally. All code is run on the server, in parallel for several clients.
- The thin client only runs the user interface!

Advantages:
- All those of network computers but the computer at the user side is even simpler (cheaper).

☞ Strong servers are needed!

**Mobile Devices**

☞ Mobile devices are hardware, computing components that move (together with their software) between physical locations.
- This is opposed to software agents, which are software components that migrate.
- Both clients and servers can be mobile (clients more frequently).

☞ Particular problems/issues:
- Mobility transparency: clients should not be aware if the server moves (e.g., the server keeps its Internet address even if it moves between networks).
- Problems due to variable connectivity and bandwidth.
- The device has to explore its environment:
  - Spontaneous interoperation: associations between devices (e.g. clients and servers) are dynamically created and destroyed.
  - Context awareness: available services are dependent on the physical environment in which the device is situated.

**Interaction Models**

How do we handle time? Are there time limits on process execution, message delivery, and clock drifts?

- Synchronous distributed systems
- Asynchronous distributed systems

**Synchronous Distributed Systems**

Main features:
- Lower and upper bounds on execution time of processes can be set.
- Transmitted messages are received within a known bounded time.
- Drift rates between local clocks have a known bound.

Important consequences:
1. In a synchronous distributed system there is a notion of global physical time (with a known relative precision depending on the drift rate).
2. Only synchronous distributed systems have a predictable behaviour in terms of timing. Only such systems can be used for hard real-time applications.
3. In a synchronous distributed system it is possible and safe to use timeouts in order to detect failures of a process or communication link.

☞ It is difficult and costly to implement synchronous distributed systems.
Asynchronous Distributed Systems

Many distributed systems (including those on the Internet) are asynchronous.

- No bound on process execution time (nothing can be assumed about speed, load, reliability of computers).
- No bound on message transmission delays (nothing can be assumed about speed, load, reliability of interconnections).
- No bounds on drift rates between local clocks.

Important consequences:

1. In an asynchronous distributed system there is no global physical time. Reasoning can be only in terms of logical time (see lecture on time and state).
2. Asynchronous distributed systems are unpredictable in terms of timing.
3. No timeouts can be used.

Asynchronous Distributed Systems (cont’d)

Asyncrhonous systems are widely and successfully used in practice.

In practice timeouts are used with asynchronous systems for failure detection. However, additional measures have to be applied in order to avoid duplicated messages, duplicated execution of operations, etc.

Fault Models

What kind of faults can occur and what are their effects?

- Omission faults
- Arbitrary faults
- Timing faults

Faults can occur both in processes and communication channels. The reason can be both software and hardware faults.

Fault models are needed in order to build systems with predictable behaviour in case of faults (systems which are fault tolerant).

Of course, such a system will function according to the predictions, only as long as the real faults behave as defined by the “fault model”. If not ......

These issues will be discussed in some of the following chapters and in particular in the chapter on “Recovery and Fault Tolerance”.

Omission Faults

A processor or communication channel fails to perform actions it is supposed to do. This means that the particular action is not performed!

- We do not have an omission fault if:
  - An action is delayed (regardless how long) but finally executed.
  - An action is executed with an erroneous result.

With synchronous systems, omission faults can be detected by timeouts.

- If we are sure that messages arrive, a timeout will indicate that the sending process has crashed. Such a system has a fail-stop behaviour.
**Arbitrary (Byzantine) Faults**

- This is the most general and worst possible fault semantics.

  Intended processing steps or communications are omitted or/and unintended ones are executed. Results may not come at all or may come but carry wrong values.

**Timing Faults**

- Timing faults can occur in synchronous distributed systems, where time limits are set to process execution, communications, and clock drifts.

  A timing fault occurs if any of this time limits is exceeded.

**Summary**

- Models can be used to provide an abstract and simplified description of certain relevant aspects of distributed systems.

- Architectural models define the way responsibilities are distributed among components and how they are placed in the system.

  We have studied three architectural models:
  1. Client-server model
  2. Peer-to-peer
  3. Several variations of the two

- Interaction models deal with how time is handled throughout the system.

  Two interaction models have been introduced:
  1. Synchronous distributed systems
  2. Asynchronous distributed systems

- The fault model specifies what kind of faults can occur and what their effects are.

  Fault models:
  1. Omission faults
  2. Arbitrary faults
  3. Timing faults