Fundamental Distributed System Models

Adapted from Distributed Systems: Concepts and Design, Coulouris, Dollimore and Kindberg
Edition 4, © Addison-Wesley 2005

Fundamental Models

**Fundamental Models** – Description of properties that are present in all distributed architectures.

- **Interaction Models** – Issues dealing with the interaction of process such as performance and timing of events.
- **Failure Models** – Specification of faults that can be exhibited by processes and communication channels.
- **Security Models** – Threats to processes and communication channels
Interaction Model Challenges

- Distributed systems are composed of cooperating processes:
  - Generally unpredictable timing and rate of message transmission delivery between processes.
  - Processes maintain their own private state
  - There is no single global notion of time.
    - Clock drift rate – relative difference process clocks differ from a perfect reference clock.

Performance Considerations

- **Latency** – delay between the transmission start of a message and its receipt.
  - Time taken by the actual message transmission will vary with load of message traffic and time required by OS services to process messages.
- **Bandwidth** – amount of information that can be transmitted over a computer network in a given amount of time.
Two Interaction Model Flavours

- **Synchronous Model**
  - Boundaries known for time to execute step, message transmission and clock drift rate.
  - Timeouts typically used to detect failures
  - Ineffective resource sharing

- **Asynchronous Model**
  - No assumptions made on process time, message transmission or clock drift.
  - Event ordering cannot be dependent on time
  - More opportunity for resource sharing, but much more complex design.

Real-time ordering of events

- Delivery time of messages cannot be predicted since clocks cannot be perfectly synchronized across a distributed system
- Logical ordering – uses logical time to provide an order among events generated by separate processes without depending on clocks.
Failure Models

- Omission Failures
  - Process – halted (crashed) and will not execute any further. “Fail-stop” if other processes can detect this state.
  - Communication – messages are “dropped” between sender and receiver (sender buffer, receiver buffer or communication channel)

- Arbitrary (Byzantine) Failures
  - Absence of process and communication omission failures, but integrity of data or processing steps is in error and undetectable by other processes.
  - Communication failures more unlikely since they are typically detected at the lower levels of message transmission software and hardware.

Processes and channels
### Omission and Arbitrary Failures

<table>
<thead>
<tr>
<th>Class of failure</th>
<th>Affects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail-stop</td>
<td>Process</td>
<td>Process halts and remains halted. Other processes may detect this state.</td>
</tr>
<tr>
<td>Crash</td>
<td>Process</td>
<td>Process halts and remains halted. Other processes may not be able to detect this state.</td>
</tr>
<tr>
<td>Omission</td>
<td>Channel</td>
<td>A message inserted in an outgoing message buffer never arrives at the other end’s incoming message buffer.</td>
</tr>
<tr>
<td>Send-omission</td>
<td>Process</td>
<td>A process completes a send, but the message is not put in its outgoing message buffer.</td>
</tr>
<tr>
<td>Receive-omission</td>
<td>Process</td>
<td>A message is put in a process’s incoming message buffer, but that process does not receive it.</td>
</tr>
<tr>
<td>Arbitrary (Byzantine)</td>
<td>Process or channel</td>
<td>Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.</td>
</tr>
</tbody>
</table>

### Timing failures

<table>
<thead>
<tr>
<th>Class of Failure</th>
<th>Affects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>Process</td>
<td>Process’s local clock exceeds the bounds on its rate of drift from real time.</td>
</tr>
<tr>
<td>Performance</td>
<td>Process</td>
<td>Process exceeds the bounds on the interval between two steps.</td>
</tr>
<tr>
<td>Performance</td>
<td>Channel</td>
<td>A message’s transmission takes longer than the stated bound.</td>
</tr>
</tbody>
</table>

- Applicable in synchronous systems with set time limits
- Not applicable in asynchronous systems since no time limits can be guaranteed.
Masking Failures

- A service masks a failure by hiding it or converting it into a more acceptable type of failure.
- System is said to be “fault tolerant”
- Examples - checksums, message retransmission, redundant servers
- Reliable communication (despite omission failures):
  - Validity – any message is eventually delivered
  - Integrity – message received is identical to message sent, and no messages are delivered twice

Failure Models

Objects and Principles

- Access rights – who is allowed to perform operations on the objects
- Principle – authorities that possess access rights (user or process)
Security Threats

- Enemy (adversary) legitimate or unauthorized connection to network.
- Threats to processes – client or server cannot determine identity.
- Threats to communication channels – enemy can copy, alter or inject messages.
- Denial of service – excessive requests with the intention of overloading resources.
- Mobile Code – Trojan horse attachments, viruses.

The enemy

Process $p$ $m$ Communication channel $m'$ Process $q$

Copy of $m$
Defeating Security Threats

- Encryption of messages using cryptography
  - secret key pairs
- Authentication of message senders
- Secure channels – service layer built on top of existing communication services
  - SSL – secure socket layer
- Make worst-case assumptions during design.

Secure channels

Principal A

Process p

Secure channel

Process q

Principal B