System Models for Distributed Systems

INF5040/9040 Autumn 2015
Lecturer: Amir Taherkordi (ifi/UiO)

August 31, 2015

Outline

1. Introduction
2. Physical Models
3. Architectural Models
4. Fundamental Models
System Models for DS

1. Introduction

System Models

- **Purpose:**
  - To illustrate/describe *common properties and design choices* for distributed systems in a *single descriptive model*

- **Three types of models**
  - **Physical models:** capture the hardware composition of a system in terms of computers and other devices and their interconnecting network
  - **Architectural models:**
    - *software architecture:* the main components of the system + their roles + how they interact
    - *system architecture:* how they are deployed in an underlying network of computers
  - **Fundamental models:** formal description of the properties that are common to architecture models. Three fundamental models:
    - interaction models, failure models and security models

---

### Physical Models

<table>
<thead>
<tr>
<th>Distributed Systems</th>
<th>Early</th>
<th>Internet-scale</th>
<th>Contemporary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale</td>
<td>Small (10-100)</td>
<td>Large</td>
<td>Ultra-large</td>
</tr>
<tr>
<td>Heterogeneity</td>
<td>Limited (typically relatively homogeneous configurations)</td>
<td>Significant in terms of platforms, languages and middleware</td>
<td>Added dimensions introduced including radically different styles of architecture</td>
</tr>
<tr>
<td>Openness</td>
<td>Not a priority</td>
<td>Significant priority with rage of standards introduced</td>
<td>Major challenge with existing standards: not yet able to embrace complex systems</td>
</tr>
<tr>
<td>Quality of Service</td>
<td>Not a priority</td>
<td>Significant priority with range of services introduced</td>
<td>Major challenge with existing services: not yet able to embrace complex systems</td>
</tr>
</tbody>
</table>
System Models for DS

Architectural Models

- To master the complexity of distributed systems, it is crucial that they are properly organized.
- Concern the logical organization of distributed systems into:

1. Communicating entities
   - Objects
   - Components
   - Web services

2. Communication paradigms
   - Interprocess communication
   - Remote invocation
   - Indirect communication

3. Roles and responsibilities
4. Placement Strategies

Indirect Communication Example

- Distributed shared data space
  - Information sharing through a database-like distributed system called MIDAS Data Space
  - Implementation challenges:
    - Availability
    - Fault-tolerance
    - Scalability
    - Consistency
    - Efficiency
Roles and Responsibilities

- Component view of client-server model

- Peer-to-peer

Placement Strategies - 1

- Multiple server processes:
  - service realized as a number of server-processes
  - several access points
Placement Strategies - 2

- Client/server model with proxy-server:
  - Cache: stores recently-used data objects that are closer to the client than the original objects themselves.
  - Proxy server: cache that is shared between several clients

Placement Strategies - 3

- Mobile code (applets)
  - Enables e.g., “push-model”: the server invokes the client, or more advanced user interfaces
Mobile agents:
- Program (code + data) that migrates between computers and executes a task on behalf of someone.

Architectural Patterns – 1
- Build on more primitive architectural elements
  - Recurring structures that have been shown to work well
  - Layering Architecture
  - Tiered Architecture
  - Thin Clients (Cloud Clients)
  - Among other patterns: Proxy, Brokerage and Reflection
Architectural Patterns – 2

Layered

Tiered

System Models for DS | 3. Architectural Models

Middleware Solutions

- Support to architectural models

- Categories:
  - Distributed Objects, Distributed Components, Publish-subscribe, Message queues, Web services, Peer-to-peer

- Limitations:
  - Dependability aspects
  - End-to-end argument
  - Context-aware and adaptive solutions
Thin Clients
- Move complexity away from end-user devices

For example:
- Virtual Network Computing (VNC): graphical desktop sharing system to remotely control another computer

Properties shared by all architecture models
- **communicates** by sending messages across a network
- requirements of **performance**, reliability, and security

Fundamental models
- abstracts over unnecessary details
- used to address questions like
  - what are the most **important entities** in the system?
  - how do they **interact**?
  - what are the characteristics that affect their individual and collective **behaviour**?

The purpose of fundamental models
- to make explicit all **relevant assumptions** about the modeled system
- to find out what is generally **feasible and not feasible** under the given assumptions
System Models for DS 4. Fundamental Models

Fundamental Models

- Aspects of distributed systems we want to express
  - Interaction model
    - processes, messages, coordination (synchronization and ordering)
    - must reflect that messages are subject to delays, and that delay limits exact coordination and maintenance of global time
  - Failure model
    - defines and classifies failures that can occur in a DS
    - basis for analysis of effects of failures and for design of fault-tolerant systems
  - Security model
    - defines and classifies security attacks that can occur in a DS
    - basis for analysis of threats to a system and for design of systems that are able to resist them


Significant Factors

- Performance of communication:
  - Latency – delay between the start of the transmission and the beginning of reception
  - Bandwidth – Total amount of information that can be transmitted
  - Jitter – Variation in the time taken to deliver a series of messages: relevant for multimedia data
- Computer Clocks:
  - Each computer: its own clock
  - Two processes running on different computers: timestamps?
  - Even reading at the same time: different timestamps!
  - Clock drift: rate for deviation from reference clock
  - How to correct time: from GPS or reference computer in the network
### System Models for DS 4. Fundamental: Interaction Models

#### Two Variants

- **Synchronous** distributed systems
  - the time to execute each step of a process: known lower and upper bounds
  - each message transmitted over a channel is received within a known bounded time
  - local clock’s drift rate from real time has a known bound

- **Asynchronous** distributed systems
  - the time to execute each step of a process can take arbitrarily long
  - each message transmitted over a channel can be received after an arbitrarily long time
  - local clock’s drift rate from real time can be arbitrarily large

### System Models for DS 4. Fundamental: Timing

#### Significance of Syn. vs Asyn. DS

- Many coordination problems have a solution in synchronous distributed systems, but not in asynchronous
  - e.g., “The two army problem” or “Agreement in Pepperland” (see Coulouris)
- Often we assume synchrony even when the underlying distributed system in essence is asynchronous
  - Internet is in essence asynchronous but we use timeouts in protocols over Internet to detect failures
  - based on estimates of time limits
  - but: design based on time limits that can not be guaranteed, will generally be unreliable
System Models for DS 4. Fundamental: Timing

Ordering of Events

- distributed coordination protocols have a need for ordering of events in time ("happened before"-relationship)
  - events: sending and receiving messages
  - example: update of replicated data must generally be done in the same order in all replica
- difficult to use physical clocks in computers for coordination (e.g., clock values in messages)
  - have limited time resolution and ticks with different rates (clock drift)
  - basic properties of message exchange limit the accuracy of the synchronization of clocks in a DS [Lamport 78]

Example: E-mail Exchange

- Example: e-mail exchange

![Diagram of e-mail exchange]

- Time line with events: send(m), rcv(m), rcv(Re:m), rcv(Re:re:m)
- Nodes X, Y, Z, A with messages m1, m2, m3
- Timing arrows indicating message flow and timing

[Diagram not visible in text format]
Logical Clocks

- Possible to describe logical ordering of events even without accurate clocks by using logical clocks
- Principle
  - If two A and B happen in the same process, then they occur in the same order: $A \rightarrow B$
  - If $A$ is sending of a message by one process and $B$ is the receipt of the same message by another process, then $A \rightarrow B$
- Happened-before relationship
  - Is derived by generalizing the two relationships above such that if $A$, $B$ and $C$ are events and $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$
- Logical clocks extends the idea above
  - More later in the course

A Failure Model

- Is a definition of in which way failures may occur in distributed systems
- Provides a basis for understanding the effects of failures
- Definition of the failure model of a service enables construction of a new service that hides the faulty behavior of the service it builds upon
  - Example: TCP on top of IP
    - TCP: reliable byte-stream service
    - IP: unreliable datagram service
Specification of a Failure Model

- Specification of failure models requires a way to describe failures.
- One approach is to classify failure types (Cristian, 1991) (Hadzilacos & Toueg, 1994)
  - Omission failures
  - Arbitrary failures
  - Timing failures
- System model:

```
  Process p
    Send m
    communication channel
  outgoing message buffer

  Process q
    Receive m
    incoming message buffer
```

Omission Failures

- A process or channel fails to perform actions that it is supposed to do.

<table>
<thead>
<tr>
<th>Failure class</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 in the other end’s incoming buffer.</td>
</tr>
<tr>
<td>Send omission</td>
<td>Process</td>
<td>A process completes a send-operation, but the message is not put into the outgoing message buffer.</td>
</tr>
<tr>
<td>Receive-omission</td>
<td>Process</td>
<td>A message is put into a process’s incoming message buffer, but the process does not receive it.</td>
</tr>
</tbody>
</table>
Arbitrary failures (Byzantine failures)

- **Process** or **channel** may exhibit **arbitrary behavior** when failing,
  - send/receive arbitrary messages at arbitrary intervals
  - a process may halt or perform “faulty” steps
  - a process may omit to respond now and then
- By adopting a byzantine failure model, we can attempt to make systems that are “**ultra-reliable**” (handles HW failures, and provide guaranteed response times)
  - control systems in air planes
  - patient monitoring systems
  - robot control systems
  - control systems for nuclear power plants

Timing Failures

- Applicable in synchronous distributed systems
  - responses that are not available to clients in a specified time interval
  - timing guarantees requires guaranteed access to resources when they are needed
- Examples:
  - control and monitoring systems, multimedia systems

<table>
<thead>
<tr>
<th>Failure class</th>
<th>Effects</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 processing steps</td>
</tr>
<tr>
<td>Performance</td>
<td>Channel</td>
<td>A message’s transmission takes longer than the stated bounds</td>
</tr>
</tbody>
</table>
Masking Failures

- Masking a failure by
  - hiding it all together or
    - e.g., message retransmission: hiding omission failures
  - converting it into a more acceptable type of failure
    - e.g., checksums for masking corrupted messages: in fact an arbitrary failure => an omission failure

- Reliable 1-to-1 communication
  - To mask some communication omission failures
  - Defined in terms of:
    - **Validity** – Any message in the outgoing message buffer is eventually delivered to the incoming message buffer
    - **Integrity** – The message received is identical to the one sent, and no messages are delivered twice
  - Threats:
    - Retransmission with no duplicate detection
    - Malicious injection of messages

Summary

- Three types of system models
  - **Physical models**: capture the hardware composition of a system in terms of computers and other devices and their interconnecting network
  - **Architecture models**: defines the components of the system, the way they interact, and the way they are deployed in a network of computers
    - Architectural elements (entities, communication paradigms)
    - Architectural patterns (layering, tiered)
    - Middleware solutions
  - **Fundamental models**: formal description of the properties that are common to all architecture models
    - interaction models
    - failure models
    - Security models (not covered in this course)