Introduction to Distributed Systems

Introduction

Benoît Garbinato
distributed object programming lab

Distributed systems

"As long as there were no ~~machines~~ networks, distributed programming was no problem at all; when we had a few weak ~~computers~~ networks, distributed programming became a mild problem and now that we have gigantic ~~computers~~ networks, distributed programming has become an equally gigantic problem. In this sense the electronic industry has not solved a single problem, it has only created them - it has created the problem of using its products."

Edgster Dijkstra, The Humbel Programmer.
Communication of the ACM, vol. 15, no. 10.
October 1972. Turing Award Lecture.

Historical background

- Hardware became continuously cheaper
- Cheap and fast networks emerged
- The example of Unix:

1969	K. Thompson & D. Ritchie develop Unix as a multi-users system on PDP-7
1979	B. Joy enhances Unix with interprocess communication facilities (BSD Unix)
1980's	Sun Microsystems used BSD Unix as operating systems for its workstations

Approach of this course (1)

- This course teaches distributed systems from both a practical and a theoretical perspective

"In theory, there is not difference between theory & practice. In practice, there is."

- The practitioner needs the theoretical perspective to understand the implicit assumptions hidden in the technologies, and their consequences
- The theoretician needs the practical perspective to validate that theoretical models, problems & solutions work in accordance to existing technologies

Approach of this course (2)

To achieve this, we will approach distributed systems through four complementary views:

- The model view
- The interaction view
- The architecture view
- The algorithm view

The model view

- What distributed entities?
  E.g., processes, objects, threads, etc.
- What time assumptions?
  E.g., synchronous, asynchronous, etc.
- What failure assumption?
  E.g., crash-stop, malicious, etc.

The interaction view

- What interaction paradigm?
  E.g., message passing, shared memory, etc.
- What reliability guarantees?
  E.g., best-effort, reliable, secure, etc.

The architecture view

- What level of decentralization?
  E.g., client/server, multi-tier, etc.
- What level of separation of concerns?
  E.g., library-based, container-based, etc.

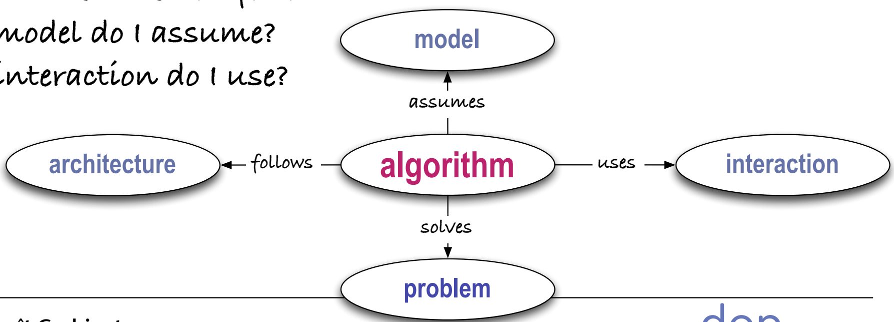
The algorithm view

- What problem?
  E.g., internet payment, consensus, etc.
- What algorithm?
  E.g., two phase commit, sliding window, etc.
- What complexity and what performance?
  E.g., NP-complete, polynomial, etc.

The big picture

When implementing a distributed program, you will always end up writing some algorithm. In doing so, you will have to answer the following questions:

- What problem am I trying to solve?
- What architecture do I follow?
- What model do I assume?
- What interaction do I use?



Layered abstractions (1)

- Sometimes, the system you are building is (yet) another abstraction level to ease the programming distributed applications. E.g., middleware, transactional monitor, etc.
- In this case, your problem is expressed in terms of the interaction you want to provide at your level.
- To avoid confusion, you thus have to clearly identify your origin & your target interactions, models and architectures, respectively.

Layered abstractions (2)

Assume you want to devise an algorithm implementing remote procedure calls

- Target interaction: remote procedure call
- Target model: partially synchronous crash-stop
- Target architecture: client/server (middleware-level)

- Origin interaction: unreliable message passing (e.g., UDP)
- Origin model ⇔ Target model
- Origin architecture: peer-to-peer (os-level)

Technologies in this course

- The Java Programming platform
- Internet protocols (TCP, UDP)
- Unix (Linux, Mac OS X, etc.) or Windows

Content & calendar

<table>
<tr><th></th><th>10:00 - 12:00</th><th>12:00 - 13:00</th></tr>
<tr><td>March 14</td><td>Introduction | Java in a nutshell</td><td>Get familiar with Java & lab tools</td></tr>
<tr><td>March 21</td><td rowspan="2">Concurrent Programming</td><td rowspan="3">Concurrent Chat</td></tr>
<tr><td>March 28</td></tr>
<tr><td>April 4</td><td rowspan="2">Remote Method Invocation</td></tr>
<tr><td>April 11</td><td rowspan="3">RMI Chat</td></tr>
<tr><td>April 25</td><td rowspan="2">Asynchronous Messaging</td></tr>
<tr><td>May 2</td></tr>
<tr><td>May 9</td><td rowspan="2">Network Programming</td><td rowspan="3">JMS Chat</td></tr>
<tr><td>May 16</td></tr>
<tr><td>May 23</td><td rowspan="2">Distributed Algorithms</td></tr>
<tr><td>May 30</td><td rowspan="4">Broadcast Chat</td></tr>
<tr><td>June 6</td><td>Tales from the academic world</td></tr>
<tr><td>June 13</td><td>Tales from the real world</td></tr>
<tr><td>June 20</td><td>Q & A</td></tr>
</table>

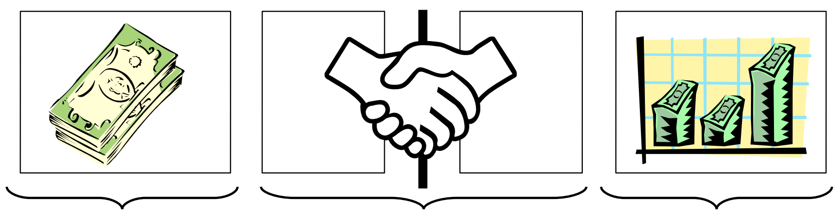
Legend: *Course* (yellow) | *Exercise* (green)

Course form

- Each Tuesday:
  - from 10 to 12 : principles
  - from 12 to 13 : exercises
- Evaluation :
  - Written final exam
  - Bonus based on exercises

Exercises

You will start from a concurrent application and you will distribute it using various programming abstractions, e.g., remote method invocations, sockets, message-oriented middleware, etc.



For further information

- http://lpdwww.epfl.ch/teaching/ids.html
- sebastien.baehni@epfl.ch
- bastian.pochon@epfl.ch
- newsgroup: epfl.ssc.ids
- benoit.garbinato@unil.ch
- http://www.hec.unil.ch/dop

Questions?