Introduction to Distributed Systems

Introduction

Benoît Garbinato
distributed object programming lab

Distributed systems (1)

"A distributed system is one that stops you from getting any work done when a machine you've never even heard of crashes."

L. Lamport, quoted by S. Müllender in Distributed Systems. 2nd edition. Addison-Wesley, 1993.

Distributed systems (2)

"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.

Distributed systems (3)

" A distributed system is a collection of autonomous computers linked by a network, with software designed to produce an integrated computing facility."

In Distributed Systems: Concept and Design.
2nd edition. Addison-Wesley, 1994.

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 <u>four complementary views</u>:

- ☐ 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.

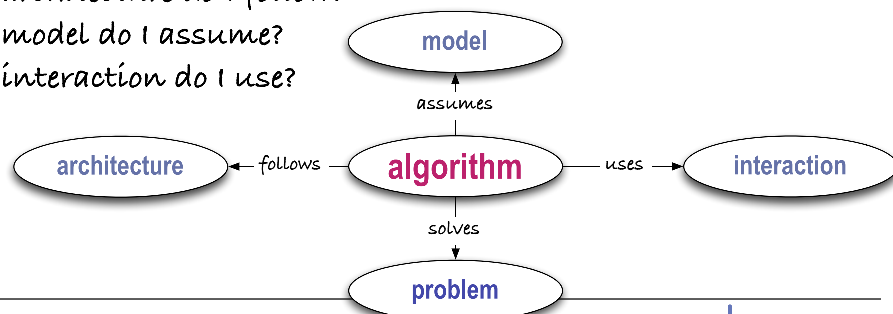
The algorithm view

- 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

	10:00 - 12:00	12:00 - 13:00
March 13	Introduction | Java in a nutshell	Get familiar with Java & lab tools
March 20	Concurrent Programming	Concurrency Exercise
March 27		Client/Server Game
April 3	Remote Method Invocation	
April 17		
April 24		
May 1	Project Presentations	Project Presentations
May 8	Network Programming	Peer-to-Peer Game Basic solution based on unreliable multicast (udp)
May 15		
May 22	Distributed Algorithms	
May 29		Peer-to-Peer Game Advanced solution based on reliable + ordered multicast
June 5	Tales from the academic world	
June 12	Tales from the real world	
June 19	Project Presentations	Project Presentations

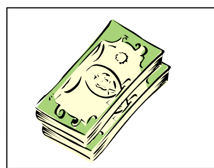
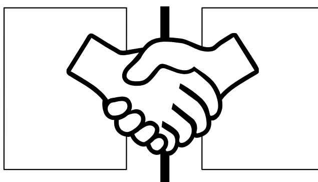
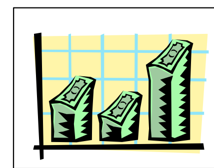
Organisation

- Each Tuesday:
  - from 10:15 to 12:00 : principles
  - from 12:15 to 13:00 : exercises
- Evaluation :
  - Exercises (E) - written assignments, compulsory
  - Final test (T) - oral exam, compulsory

If $T \geq 3$: Final grade $= 0.4 \times E + 0.6 \times T$
If $T < 3$: Final grade $= T$

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.

client-side business logic | programming abstractions | server-side business logic

Further information

- http://mtc.epfl.ch/courses/IDS-2007
- vasu.singh@epfl.ch
- benoit.garbinato@unil.ch

Questions?