Chapter 1: Distributed Systems: What is a distributed system?

Fall 2013
Course Goals and Content

■ Distributed systems and their:
  ■ Basic concepts
  ■ Main issues, problems, and solutions
  ■ Structured and functionality

■ Content:
  ■ Distributed systems
    - Architectures, goal, challenges
    - Where our solutions are applicable
  ■ Synchronization: Time, coordination, decision making
  ■ Replicas and consistency
  ■ Fault tolerance
  ■ Large-scale distributed systems in real world
Course Material

  - 2002 edition also ok
- Coulouris, Dollimore, Kindberg: Distributed Systems, Concepts and Design; Addison-Wesley 2005
- Selected content from Barroso L. A. and Hölzle U.: The Datacenter as a Computer (online, see course website)

- Lecture slides on course website
  - NOT sufficient by themselves
  - Help to see what parts in book are most relevant
  - On some topics, slides cover more material than the book
Course Exams

- Normal way (recommended)
  - Exercises, home exercises, course exam

- Grading:
  - Exam 42 points, date December 11, 2013
  - Exercises 12 points (6 exercises, scaled to 0—12)
  - Home exercises 12 points (4 exercises)
  - Grading based on 60 point maximum
  - Need 30 points to pass with minimum 16 points in exam
  - 50 points will give a 5

- Possible to take as separate exam
Exercises

- Bi-weekly exercises:
  - Smaller assignments
  - Exception to schedule: 29.11. has exercise, 6.12. does not
  - Complete beforehand, present your work at exercise sessions to gain points
  - Must make an effort to get points!

- Home exercises
  - 4 larger exercises
    - 2 writing exercises
    - 2 programming exercises
  - Due dates will be announced later, deadlines fixed!
People

- Jussi Kangasharju
  - Lectures:
    - Period I: Mon 10-12 and Thu 10-12 in C222
    - Period II: Mon 10-12 and Thu 10-12 in D122
    - 12 lectures, usually on Mondays
    - Thursday sessions workshops/discussion/help/…
  - Exercise group: Fri 12-14 in B222 (bi-weekly)
  - Office hour: Mon 13-14 or ask for appointment by email

- Ossi Karkulahti
  - Exercise group: Fri 12-14 in B222 (bi-weekly)
  - Home exercises
  - Office hour: During exercises or ask appointment by email
Lectures vs. Workshops

- Lectures on Mondays present new material
  - There can and should be discussion as well 😊

- Workshops on Thursdays are for discussion and help
  - First three workshops mainly tutorials for new students
  - Later workshops discussion about exercises
  - Every week has some scheduled program
  - Program posted on website in advance
Questions?
Chapter Outline

- Defining distributed system
- Examples of distributed systems
- Why distribution?

- Goals and challenges of distributed systems

- Where is the borderline between a computer and a distributed system?
- Examples of distributed architectures
Definition of a Distributed System

A distributed system is

a collection of independent computers
that appears to its users
as a single coherent system.

... or ...
as a single system.
Where Does the Definition Leave Us?

Which of the following are distributed systems?

- Multi-core processor
- Parallel systems
- Multi-processor computer
- One data center
- Web
- Internet
- Computing cluster
- Corporate intranet
- Local Area Network
- Network of data centers
What About the Following?

- Car
- My home theater setup
- Ticket reservation system
- Nuclear power plant
- Mobile phone
- Factory
- My office
- Airplane
Examples of Distributed Systems

- one single “system”
- one or several autonomous subsystems
- a collection of processors => parallel processing
  => increased performance, reliability, fault tolerance
- partitioned or replicated data
  => increased performance, reliability, fault tolerance
Why Distribution?

Sharing of information and services

Possibility to add components improves availability

reliability, fault tolerance

performance

Leads to scalability

And a large set of gotchas…
Goals and challenges for distributed systems
Goals

- Making resources accessible
- Distribution transparency
- Openness
- Scalability
- Security
- System design requirements
Challenges for Making Resources Accessible

- Naming
- Access control
- Security
- Availability
- Performance
- Mutual exclusion of users, fairness
- Consistency in some cases
Challenges for Transparency

- The fundamental idea: a collection of independent, autonomous actors

- Transparency:
  - Concealment of distribution
  - => user’s point of view: a single unified system
## Transparencies

<table>
<thead>
<tr>
<th>Transparency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Hide differences in data representation and how a resource is accessed</td>
</tr>
<tr>
<td>Location</td>
<td>Hide where a resource is located (*)</td>
</tr>
<tr>
<td>Migration</td>
<td>Hide that a resource may move to another location (*) (the resource does not notice)</td>
</tr>
<tr>
<td>Relocation</td>
<td>Hide that a resource may be moved to another location (*) while in use (the others don’t notice)</td>
</tr>
<tr>
<td>Replication</td>
<td>Hide that a resource is replicated</td>
</tr>
<tr>
<td>Concurrency</td>
<td>Hide that a resource may be shared by several competitive users</td>
</tr>
<tr>
<td>Failure</td>
<td>Hide the failure and recovery of a resource</td>
</tr>
<tr>
<td>Persistence</td>
<td>Hide whether a (software) resource is in memory or on disk</td>
</tr>
</tbody>
</table>

(*) Note the various meanings of ”location”: network address (several layers) ; geographical address
Challenges for Transparencies

- Replications and migration cause the need for ensuring consistency and distributed decision-making.
- Failure modes.
- Concurrency.
- Heterogeneity.
### 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 <em>send</em>, 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>
Failure Handling

- More components => increased fault rate
- Increased possibilities
  - more redundancy => more possibilities for fault tolerance
  - no centralized control => no fatal failure

Issues
- Detecting failures
- Masking failures
- Recovery from failures
- Tolerating failures
- Redundancy

New: partial failures
New: warehouse-scale computing (we return to this)
Concurrency

- Several simultaneous users => integrity of data
  - mutual exclusion
  - synchronization
  - ext: transaction processing in data bases
- Replicated data: consistency of information?
- Partitioned data: how to determine the state of the system?
- Order of messages?

- There is no global clock!


Consistency Maintenance

- Update ...
- Replication ...
- Cache ...
- Failure ...
- Clock ...
- User interface ....

... consistency
Heterogeneity

- Heterogeneity of
  - networks
  - computer hardware
  - operating systems
  - programming languages
  - implementations of different developers

- Portability, interoperability
- Mobile code, adaptability (applets, agents)
- Middleware (CORBA etc)
- Degree of transparency? Latency? Location-based services?
Challenges for Openness

- Openness facilitates
  - interoperability, portability, extensibility, adaptivity

- Activities addresses
  - extensions: new components
  - re-implementations (by independent providers)

- Supported by
  - public interfaces
  - standardized communication protocols
Challenges for Scalability

Scalability:

- The system will remain effective when there is a significant increase in:
  - number of resources
  - number of users

- The architecture and the implementation must allow it
- The algorithms must be efficient under the circumstances to be expected
  - Example: the Internet
Challenges: Scalability (cont.)

- Controlling the cost of physical resources
- Controlling performance loss
- Preventing software resources running out
- Avoiding performance bottlenecks
- Mechanisms (implement functions) & Policies (how to use the mechanisms)

Scaling solutions

- Asynchronous communication, decreased messaging
- Caching (all sorts of hierarchical memories: data is closer to the user → no wait / assumes rather stable data!)
- Distribution i.e. partitioning of tasks or information (domains) (e.g., DNS)
Challenges for Security

- Mostly similar to normal challenges in wide-area networks
  - Sometimes easier (closed, dedicated systems)

- Solution techniques
  - Cryptography
  - Authentication
  - Access control techniques

- Policies
  - Access control models
  - Information flow models

- Leads to: secure channels, secure processes, controlled access, controlled flows
Environment challenges

- A distributed system:
  - HW / SW components in different nodes
  - Components communicate (using messages)
  - Components coordinate actions (using messages)

- Distances between nodes vary
  - In time: from msecs to weeks
  - In space: from mm’s to Mm’s
  - In dependability

- Autonomous independent actors (=> even independent failures!)

  No global clock

  Global state information not possible
Challenges: Design Requirements

- Performance issues
  - responsiveness
  - throughput
  - load sharing, load balancing
  - issue: algorithm vs. behavior

- Quality of service
  - correctness (in nondeterministic environments)
  - reliability, availability, fault tolerance
  - security
  - performance
  - adaptability
Where is the borderline between a computer and distributed system?
Hardware Concepts

- Characteristics which affect the behavior of software systems
- The platform ....
  - the individual nodes ("computer", "processor")
  - communication between two nodes
  - organization of the system (network of nodes)
- ... and its characteristics
  - capacity of nodes
  - capacity (throughput, delay) of communication links
  - reliability of communication (and of the nodes)

→ Which ways to distribute an application are feasible
Different basic organizations and memories in distributed computer systems
Multiprocessors (1)

A bus-based multiprocessor.

Essential characteristics for software design
• fast and reliable communication (shared memory) => cooperation at "instruction level" possible
• bottleneck: memory (especially the "hot spots")
General Multicomputer Systems

- Hardware: Possibly very heterogeneous
- Loosely connected systems
  - nodes: autonomous
  - communication: slow and vulnerable
  - => cooperation at "service level"
- Application architectures
  - multiprocessor systems: parallel computation
  - multicomputer systems: distributed systems
  - (how are parallel, concurrent, and distributed systems different?)
# Software Concepts

<table>
<thead>
<tr>
<th>System</th>
<th>Description</th>
<th>Main Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOS</td>
<td>Tightly-coupled operating system for multiprocessors and homogeneous multicompilers</td>
<td>Hide and manage hardware resources</td>
</tr>
<tr>
<td>NOS</td>
<td>Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)</td>
<td>Offer local services to remote clients</td>
</tr>
<tr>
<td>Middleware</td>
<td>Additional layer atop of NOS implementing general-purpose services</td>
<td>Provide distribution transparency</td>
</tr>
</tbody>
</table>

DOS: Distributed OS; NOS: Network OS
History of distributed systems

- RPC by Birel & Nelson - 84
- Network operating systems, distributed operating systems, distributed computing environments in mid-1990; middleware referred to relational databases
- Distributed operating systems – “single computer”
  - Distributed process management
    - process lifecycle, inter-process communication, RPC, messaging
  - Distributed resource management
    - resource reservation and locking, deadlock detection
  - Distributed services
    - distributed file systems, distributed memory, hierarchical global naming
History of distributed systems

- late 1990’s distribution middleware well-known
  - generic, with distributed services
  - supports standard transport protocols and provides standard API
  - available for multiple hardware, protocol stacks, operating systems
  - e.g., DCE, COM, CORBA
- present middlewares for
  - multimedia, realtime computing, telecom
  - ecommerce, adaptive / ubiquitous systems
Misconceptions tackled

- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero
- Bandwidth is infinite
- Transport cost is zero
- There is one administrator
- There is inherent, shared knowledge
Multicomputer Operating Systems (1)

General structure of a multicomputer operating system

Machine A  Machine B  Machine C

Distributed applications

Distributed operating system services

Kernel  Kernel  Kernel

Network

General structure of a multicomputer operating system
Multicomputer Operating Systems (2)

Alternatives for blocking and buffering in message passing.

Diagram showing
- Sender
- Sender buffer
- S1
- S2
- Possible synchronization point
- Receiver
- Receiver buffer
- S3
- S4
- Network

Kangasharju: Distributed Systems
10 September 2013
a) Pages of address space distributed among four machines

b) Situation after CPU 1 references page 10

c) Situation if page 10 is read only and replication is used
Distributed Shared Memory Systems (2)

False sharing of a page between two independent processes.
Network Operating System (1)

General structure of a network operating system.
Network Operating System (2)

Two clients and a server in a network operating system.
Different clients may mount the servers in different places.
Software Layers

- **Platform**: computer & operating system & ..
- **Middleware**:
  - mask heterogeneity of lower levels
  - (at least: provide a homogeneous “platform”)
  - mask separation of platform components
    - implement communication
    - implement sharing of resources
- **Applications**: e-mail, www-browsers, …
Positioning Middleware

General structure of a distributed system as middleware.
Middleware

- Operations offered by middleware
  - RMI, group communication, notification, replication, … (Sun RPC, CORBA, Java RMI, Microsoft DCOM, …)

- Services offered by middleware
  - naming, security, transactions, persistent storage, …

- Limitations
  - ignorance of special application-level requirements

**End-to-end argument:**

- Communication of application-level peers at both ends is required for reliability
In an open middleware-based distributed system, the protocols used by each middleware layer should be the same, as well as the interfaces they offer to applications.
## Comparison between Systems

<table>
<thead>
<tr>
<th>Item</th>
<th>Distributed OS</th>
<th>Network OS</th>
<th>Middleware-based OS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Multiproc.</td>
<td>Multicomp.</td>
<td></td>
</tr>
<tr>
<td>Degree of transparency</td>
<td>Very High</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Same OS on all nodes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Number of copies of OS</td>
<td>1</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Basis for communication</td>
<td>Shared memory</td>
<td>Messages</td>
<td>Files</td>
</tr>
<tr>
<td>Resource management</td>
<td>Global, central</td>
<td>Global, distributed</td>
<td>Per node</td>
</tr>
<tr>
<td>Scalability</td>
<td>No</td>
<td>Moderately</td>
<td>Yes</td>
</tr>
<tr>
<td>Openness</td>
<td>Closed</td>
<td>Closed</td>
<td>Open</td>
</tr>
</tbody>
</table>
More examples on distributed software architectures
Architectural Models

- Architectural models provide a high-level view of the distribution of functionality between system components and the interaction relationships between them.
- Architectural models define:
  - components (logical components deployed at physical nodes)
  - communication
- Criteria:
  - performance
  - reliability
  - scalability, ..
Client-Server Architectures

- General interaction between a client and a server.
The general organization of an Internet search engine into three different layers:

- **User interface level**: User interface
- **Processing level**: Query generator, HTML generator, Ranking component
- **Data level**: Database with Web pages, Web page titles with meta-information

Keyword expression leads to Query generator, which generates a list of page titles ranked by the Ranking component. This list is then used to generate an HTML page containing a list of results.
Multitiered Architectures (1)

Alternative client-server organizations.

(a) (b) (c) (d) (e)

User interface
Application
Database

Client machine

Server machine

User interface
Application
Database

User interface
Application
Database

User interface
Application
Database

User interface
Database
Multitiered Architectures (2)

Client - server: generalizations

- A client: node 1
  server: node 2

- B client: node 2
  server: node 3

the concept is related to communication not to nodes
An example of a server acting as a client.
Modern Architectures

An example of horizontal distribution of a Web service.

Front end handling incoming requests

Requests handled in round-robin fashion

Replicated Web servers each containing the same Web pages

Disks

Internet

An example of horizontal distribution of a Web service.
Chapter Summary

- Introduction into distributed systems
- Challenges and goals of distributing
- Examples of distributed systems