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

Fall 2012
Sini Ruohomaa

(Slides joint work with Jussi Kangasharju et al. Figures from course material)
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
- My workroom electronics
- Multi-processor computer
- One data center
- Web
- Internet
- Computing cluster
- Corporate intranet
- Local Area Network
- Network of data centers
Examples of Distributed Systems

Distributed application

- 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
Goals and challenges for distributed systems

Getting a feel of the playing field
Goals

- Making resources accessible
- Openness
- Scalability
- Security
- Fitting the given concrete environment
- Fulfilling system design requirements
- Distribution transparency

- What could go wrong?
Challenges for Making Resources Accessible

- **Goal:** should be easy for users to access/share resources
- **What it takes to achieve this:**
  - Naming
  - Access control
  - Security
  - Availability
  - Performance
  - Mutual exclusion of users, fairness
  - Consistency in some cases
Challenges for Openness

- Goal: follow standard rules, allow different players on field
  - Interoperability: allow different solutions to coexist
  - Portability: solution executable as is in different systems
  - Extensibility: simple to add new components, or
  - Possible to reimplement (by independent providers)

- Supported by
  - Public, well-specified interfaces
  - Standardized communication protocols
  - Separation of policy (rules of use) from mechanism (functionality available for use): allows change of policy
Challenges for Scalability (1/2)

Scalability:

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

- The architecture and the implementation must allow it
- The algorithms must be efficient under the circumstances to be expected
  - Example: the Internet
Challenges for Scalability (2/2)

- Controlling the cost of physical resources
- Controlling performance loss
- Preventing software resources running out
- Avoiding performance bottlenecks
- 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., the DNS, handling domain names on the Internet)
Challenges for Distribution Transparency (1+)

- Goal: Collection of independent, autonomous actors appear to user as single unified system
  - Hide the distribution

- Different categories of transparency:
## Transparencies (RM-ODP standard, 1998)

<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>Transaction</td>
<td>Hide that multiple competing users perform concurrent actions on the resource</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 Distribution Transparency (2)

- **Concurrency**
  - Many things happening at the same time
- **Replications and migration cause additional requirements:**
  - Ensure consistency between different replicas and
  - Support distributed decision-making
- **Heterogeneity**
  - All the differences in hardware, software, etc to account for
- **Failure models**
  - Things can go wrong in different ways
Handling Concurrency

Concurrency:
- Several simultaneous users => integrity of data
  - mutual exclusion
  - synchronization
  - ext: transaction processing in databases
- 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
Handling 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?
## Failure handling: what can go wrong?
### 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 <code>send</code>, 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>
## What can go wrong? 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
Challenges for Security

- Mostly similar to normal challenges in wide-area networks
  - Sometimes easier, with 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
Challenges from the Environment

- 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 1 millisecond to weeks
  - in space: from 1 mm to thousands of kilometers
  - in dependability: link always there or completely unreliable

- Autonomous independent actors (=> independent failures, too!)

  No global clock

  Global state information not possible
Challenges from Design Requirements

- Performance requirements
  - responsiveness
  - throughput
  - load sharing, load balancing
  - issue: abstract algorithm vs. actual system behavior

- Quality of service requirements
  - correctness (in nondeterministic environments)
  - reliability, availability, fault tolerance
  - security (again with the security!)
  - performance
  - adaptability
False assumptions everyone makes when developing their first distributed application:

- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero
- Bandwith is infinite
- Transport cost is zero
- There is one administrator
- There is inherent, shared knowledge

By Peter Deutsch (creator of Ghostscript)
Systems, Architectures and System Architectures

Where is the borderline between a computer and a distributed system?
Hardware: The Bottom Layer

- The behavior of software systems is affected by:
  - 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
Basic Organizations of a Node

Different basic organizations and memories in distributed computer systems
A Look at Hardware Level: Multiprocessors

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 setup may be very heterogeneous
- Loosely connected systems
  - Nodes are autonomous
  - Communication is slow and vulnerable
  - => Cooperation at ”service level”
- Application architectures
  - Multiprocessor systems do parallel computation
  - Multicomputer systems form distributed systems
Some concepts for the coming history tour

<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 multicomputers</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
Brief history of distributed systems (1/3)

- 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 – form ”a 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
Brief history of distributed systems (2/3)

- 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
  - Examples: Distributed Computing Environment (DCE) ’90s, Microsoft’s COM and later .NET framework, OMG’s CORBA

- present middlewares for
  - multimedia, realtime computing, telecom
  - ecommerce, adaptive / ubiquitous systems
Brief history of distributed systems (3/3)

- Late 1990’s: distribution middleware well-known
  - Generic, with distributed services
  - Support standard transport protocols and provide standard API
  - Available for multiple hardware, protocol stacks, operating systems
  - Examples: Distributed Computing Environment (DCE) ’90s, Microsoft’s COM and later .NET framework, OMG’s CORBA

- Present middlewares exist for
  - multimedia, realtime computing, telecommunications
  - eCommerce, adaptive / ubiquitous systems
## Operating systems and middleware

<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 multicomputers</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
Multicomputer Operating Systems (1)

General structure of a multicomputer operating system
Alternatives for blocking and buffering in message passing.
Distributed Shared Memory Systems (1)

(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.

File server

Disks on which shared file system is stored

Request

Reply

Network

Client 1

Client 2
Network Operating System (3)

Different clients may mount the servers in different places.
Above the Operating System: 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
  - Remote Method Invocation (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

Client-server model generalized,
Peek at architectural styles
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 and communication.
- Criteria for architecture design:
  - performance
  - reliability
  - scalability, …
Client-Server Architectures

- General interaction between a client and a server.
Layered architecture

The general organization of an Internet search engine into three different layers.

- **User interface level**
  - User interface
  - HTML page containing list

- **Processing level**
  - HTML generator
  - Ranking component
    - Ranked list of page titles

- **Data level**
  - Database with Web pages
    - Database queries
    - Web page titles with meta-information

Keyword expression

Query generator

Database with Web pages

The general organization of an Internet search engine into three different layers.
Multitiered Architectures (1)

Alternative client-server organizations.
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.
Chapter Summary

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