Distributed Systems

Introduction and System Architectures

Tenshi Hara
tenshi.hara@tu-dresden.de
https://www.rn.inf.tu-dresden.de/hara
Organisational Information

- Lecture Slides and Tutorial Material at https://www.rn.inf.tu-dresden.de/ds
- Lectures: Thursdays, 14:50h (2:50p), APB/E023
- Tutorials: Tuesdays, 09:20h (9:20a), APB/E023
  - No Tutorial on 21 November 2017
- INF-PM-ANW:
  - You must give a 15-minute presentation (exam advance)
  - request a topic latest by 19 December 2017 (talk to Thomas Springer or send an e-mail)
Outline

- Overview and Applications
- What are Distributed Systems?
- Models
- Architectures
- Overview of development models
Application Examples

- eCommerce (Online Shops, Banking, Travel Booking)
- Enterprise software
  - ERP - Enterprise Resource Planning
  - CRM - Customer Relationship Management
  - SCM - Supply Chain Management
- Teamwork (CSCW – Computer Supported Collaborative Work)
- Process control (CIM – Computer Integrated Manufacturing)
- Scientific Research, large-scale commercial applications
  (Computationally intensive problems – Grid / Cloud computing)
Motivation for Distribution

- Data, function and load distribution
- Scalability without changing components
- Decentralization and co-operation
- Locality properties and efficiency
- Integration of partial applications
- Remote resource access
- Parallelization of processes for increased efficiency
- Fault tolerance: reliability and availability
What are Distributed Systems?

“A Distributed System comprises several single components on different computers, which normally do not operate using shared memory and as a consequence communicate via the exchange of messages. The various components involved cooperate to achieve a common objective such as the performing of a business process.”

Schill & Springer
Main Characteristics of Distributed Systems

- Multiple individual components
- Spatially separated components
- Components possess own memory
- Access to common resources (e.g., databases)
- Communication via messages
- Cooperation towards a common objective
- Networked workstations, also organizationally integrated
- Super-proportionally increasing communication performance
- Application components (domain-specific) on top of infrastructure
Example System: E-Commerce

Client

Web Server

Application Server

Purchase Order Preprocessing

Web Service

Application Server

Store management

Store tables

Application Server

Administration of customer data

Customer data

Online shop

Web Server

Application Server

Web Service

Application Server

Data base

Supplier
UML Diagram of E-Commerce System
Distributed System

- Physical computer nodes (processor + storage)
- Direct / indirect computer coupling
  - local networks (Ethernet, WLAN etc.)
  - high-performance networks (Gigabit Ethernet)
  - gateways / bridges / switches
  - global radio networks (GSM, UMTS, LTE)
- Transport-oriented communication protocols (TCP/IP, UDP/IP)
- Communicating processes
  - complete logical connection
  - no complete physical connection (communication via gateways)
- System oriented resources (file system, threads, system programs)
- Distributed storage, decentralized, co-operative
- Distributed applications (domain-specific) on top of distributed systems
Distributed System Example: Infrastructure

Characteristics
- Decentralized system structure
- Heterogeneity of systems

Requirements
- Scalability
- Fault tolerance
- Security concepts
- Consistency - Transaction processing
Required bridging between heterogeneous networks, system platforms and applications

→ Middleware
**Def. of Middleware**: “Infrastructure services implementing commonly used functionality for distributed applications in a reusable way for bridging of heterogeneity of different systems and networks”
**Models: Client/Server**

- **Roles:**
  - Client - Service user (e.g., machine used by the customer)
  - Server – Service bringer (e.g., hosting company website)

- **Hierarchical System** – Server can be client to another server

- **Client** accesses functionality of server through calls
  - E.g., Remote Procedure Call

- **Synchronous communication**

- **Asynchronous communication**
- Objects of arbitrary granularity (e.g., a server or a customer record)
- Local and remote objects
- Objects can be passed using Value and Reference Parameter Semantics.
- Dynamic object migration e.g., to facilitate local communication to avoid high data traffic over the network.
- Realised through Java Remote Method Invocation and other programming languages
- Cross-Programming Language approach – transfer objects in XML
### Comparison of system models

<table>
<thead>
<tr>
<th></th>
<th>Client/Server-Model</th>
<th>Distributed Object-Oriented Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>processing model</td>
<td>procedural model with restrictions</td>
<td>object communication</td>
</tr>
<tr>
<td>data access type</td>
<td>indirect data access via RPC-Server</td>
<td>direct access to objects</td>
</tr>
<tr>
<td>data transfer</td>
<td>value parameter semantics</td>
<td>reference parameter semantics</td>
</tr>
<tr>
<td>identity</td>
<td>not system-wide unique</td>
<td>system-wide unique</td>
</tr>
<tr>
<td>granularity</td>
<td>server of coarse granularity</td>
<td>objects of any granularity</td>
</tr>
<tr>
<td>placement</td>
<td>fixed placement</td>
<td>modifiable placement</td>
</tr>
</tbody>
</table>

Higher transparency grade and improved distribution influence on distribution with object-oriented model.
- Lack of flexibility through inheritance in object oriented approach
- Components possess larger granularity than objects
- Simplified development through higher level of abstraction
- Object-oriented programming languages can be used to implement components → components can consist of many objects
Models: Component-based

- Separation of code and configuration
- Provide simplified programming interfaces for the access of system services
- Specific tools for component based software development
- Examples of component based approaches: Enterprise JavaBeans, OSGi and Microsoft .NET-Components
Models: Service-oriented

Distributed Systems – Introduction and System Architectures
Models: Service-oriented

- Higher level of abstraction than objects and components (e.g., technology independence)
- Principles of Service Oriented Architecture (SOA) and Web Services (WS): services are *offered, searched for* and *used*
- Loosely coupled; interoperability across platform and business borders
- Composition of services for complex processes
- Programming Language Independence
  - Accessible via service interfaces and SOAP
  - Web Services Description Language based on XML
Architectures: Two/Three Tier

User interface, if necessary pre-processing (thin client vs. fat client)

- **2-tier**: two-level structure (user-interface ↔ Host); simple, but less flexible
- **3-tier**: three-level structure; preferable for complex applications

**Presentation Tier**

- Client (e.g., Point of Sale)
- Client (e.g., Fieldworker)
- Client (e.g., Home Access)

**Processing Tier**

- Application Logic (Processing Tier)
  - Server (e.g., Pre-processing)

**Persistence Tier**

- Server (e.g., Customer History Database)
Architectures: Clustering

Client (e.g., Point of Sale)

Client (e.g., Fieldworker)

Client (e.g., Home Access)

Server Cluster

Load Balancing

Preprocessing

Customer Base Data

Distributed Systems – Introduction and System Architectures
Architectures: Clustering

- Replication through multiple Servers
  - Processing, Persistence and/or Application Data can be offered by Several Servers

- Advantages:
  - Load balancing
  - Fault tolerance
  - Parallel processing
  - Higher processing speed

- Challenges:
  - Consistency between replicated data
    - Complexity due to synchronisation and conflict handling
Grid - Availability and use of computing power in a way analogous to the provision of water and electricity in modern utility grids

- Single organisations may not have sufficient resources for data and computationally intensive problems
- Aggregation of computing resources from multiple organisations
- Individual organisations can access resources on demand to cater for fluctuating need
- Saving in processing time and hardware costs outweighs the rise in coordination and synchronisation complexity

Example projects: Earth System Grid, Human Genome Research
- General classes of components
- Complexity varies in the form of a sand clock
- Standardised protocols for exchanging messages
  - how resources/services are located, reserved, used & managed
  - e.g., communication layer protocols: TCP, UDP, DNS
Cloud Computing is advancement of Grid Computing

Provision and revocation of resources as disk space, computational power, network capacity and application → elasticity

SOA-Principle:
- provision of cloud infrastructure, platforms, applications as electronic services
- Composition and integration into business processes

Distinction to Grid:
- Provisioning of resources only by one provider
- Infrastructure not always decentralized
- Central administration of infrastructure
- Economic orientation: monitoring of use and quality of resources, pay-per-use
Service models:

- Cloud Infrastructure as a Service (IaaS)
  - Amazon Web Services, OpenStack
- Cloud Platform as a Service (PaaS)
  - Azure Service Platform, Google App Engine, Heroku
- Cloud Software as a Service (SaaS)
  - Google Apps, Salesforce

Implementation

- Public Cloud
- Private Cloud
- Community Cloud
- Hybrid Cloud
Architectures: Peer-to-Peer

- Decentralized architecture: direct communication between peers
- Peers act as service providers and users
- Mechanism for finding service providing peers necessary
- Application areas:
  - Parallel Applications
  - Content Management and Sharing
  - Collaboration
Classification of Peer-to-Peer Networks

unstructured P2P

- centralized P2P
  - server for coordination and search
  - example: Napster

- pure P2P
  - no centralized coordination
  - example: Gnutella

- hybrid P2P
  - dynamic central entities, some peers act as coordinator
  - example: BitTorrent

structured P2P

- Distributed Hash-Table
  - "fixed" connections in overlay
  - example: Kademlia
Summary: Technologies for Distributed Systems

Integrity

Transaction Monitors
Message Oriented Middleware

Client/Server, Remote Procedure Call (RPC) (for instance DCE - Distributed Computing Environment)

Object Transaction Monitor

RMI-/ .NET/SOAP- Object-oriented Basic comm.

Application Server / SOA / Cloud
Component Frameworks (Enterprise JavaBeans, OSGi, .NET)

Flexibility

Usability by application developer
References