TDTS04 Distributed Systems
Part A

Slides derived from "Distributed Systems: Principles and Paradigms", by Andrew S. Tanenbaum and Maarten Van Steen, Pearson Int. Ed., as well as material from other instructors including Niklas Carlsson and Juha Takkinen.

MIKAEL ASPLUND
REAL-TIME SYSTEMS LABORATORY
DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE
Linköpings universitet

Goals

- Study concepts that build the foundations of large-scale systems
- Learn about tradeoffs when building large-scale systems
- Learn from case studies, example systems
- Get exposure to system building and (if time) distributed systems research

Distributed systems

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

- Hardware view
  - Multiple independent but cooperating resources
- Software view
  - Single unified system

Distributed systems

- Why?
- Why not?

Distributed systems

- Benefits
  - Performance
  - Distribution
  - Reliability
  - Incremental growth
  - Sharing of data/resources
- Problems
  - Difficulties developing software
  - Fault management
  - Network problems
  - Security problems

Examples

Distributed systems

- Goals
  - Sharing (incl. openness and heterogeneity)
  - Transparency
  - Scalability (incl. communication)
  - Dependability

Sharing

- Multiple users can share and access remote resources
  - Hardware, files, data, etc.
- Open standardized interface
  - Often heterogeneous environment (hardware, software, devices, underlying network protocols, etc.)
  - Middleware layer to mask heterogeneity
- Separate policies from mechanisms

Transparency in a Distributed System

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that multiple copies of a resource exist
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

Different forms of transparency in a distributed system.

Scalability

- Allow the system to become bigger without negatively affecting performance
- Multiple dimensions:
  - **Size:** Adding more resources and users
  - **Geographic:** Dispersed across locations
  - **Administrative:** Spanning multiple administrative domains

Scalability

- Scalability problems appear as performance problems
  - System load, storage requirements, communication overhead, ...
- Some common techniques:
  - – Divide and conquer
  - – Replication
  - – Distributed operation
  - – Service aggregation
  - – Asynchronous communication
  - – Multicast

Dependability

- Property of a computing system which allows reliance to be justifiably placed on the service it delivers.

  [Avizienis et al. 2004]

Attributes of dependability

- **Safety**
  - non-occurrence of catastrophic consequences on the environment
- **Availability**
  - the readiness for usage
- **Reliability**
  - continuity of correct service

Attributes of dependability

- **Maintainability**
  - ability to undergo repairs and modifications.
- **Integrity**
  - non-occurrence of unauthorized alteration of information
- **Confidentiality**
  - absence of unauthorized disclosure of information

Means of achieving dependability

- **Fault prevention**
  - Design in such a way that occurrence of faults are reduced
- **Fault tolerance**
  - Design system to cope with faults
- **Fault removal**
  - Review and test system to remove faults
- **Fault forecasting**
  - Predict the occurrence of faults in order to justify the dependability of the system

Common Pitfalls

- The network is reliable
- The network is secure
- The network is homogenous
- The topology does not change
- Latency is zero
- Bandwidth is infinite
- Transport cost is zero
- There is one administrator

Distributed system architecture

- A distributed application runs across multiple machines
  - How to organize the various pieces of the application?
  - Where is the user interface, computation, data?
  - How do different pieces interact with each other?

Roles

- **Client-server**
  - Client implements the user interface
  - Server(s) has most of the functionality
    - Computation, data
    - E.g.: Web
- **Peer-to-peer (P2P)**
  - Each component is symmetric in functionality
  - Peer: Combination of server-client
  - No “well-known” centralized server
- **Hybrid**
  - Combination of the two

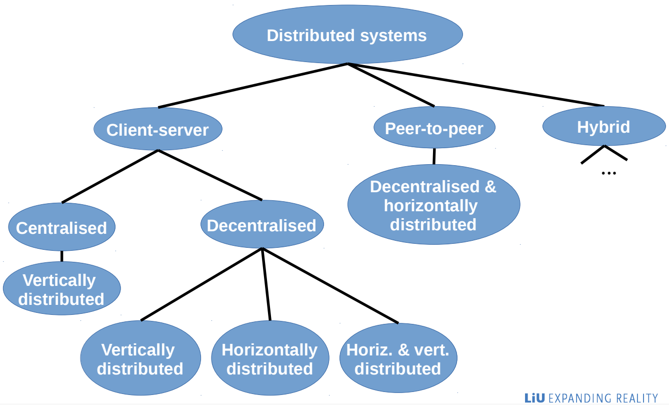
System organisation

- **Centralised**
  - Most functionality is in a single unit
- **Decentralised**
  - Functionality is spread across multiple units

Types of distribution

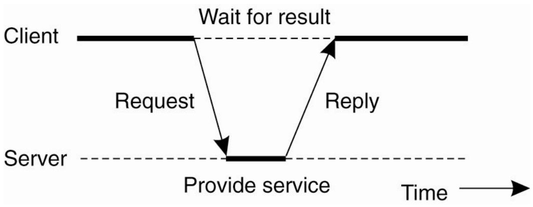
- **Vertical distribution**
  - Logically different components on different machines
  - e.g., multitiered architectures
- **Horizonal distribution**
  - Multiple logically equivalent parts
  - Potentially operating on different data

A taxonomy of architectural models



Centralized client-server architectures

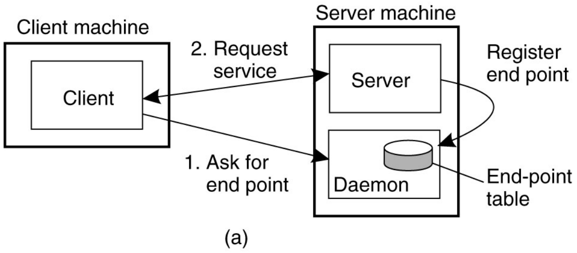
Figure 2-3. General interaction between a client and a server.



Server design issues

- **Server organization**; e.g., How to process client requests?
  - – Iterative
  - – Concurrent
    - Multithreaded
    - Fork (unix)
  - – Stateless or stateful
- **Client contact**; e.g., how to contact end point (port)
  - – Well-known
  - – Dynamic: daemon; superserver (unix)

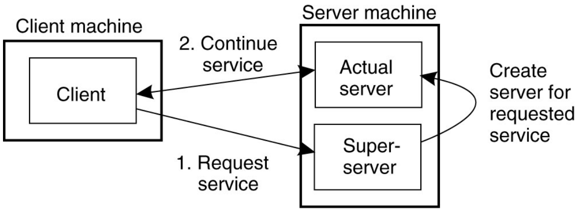
End point, general design issues



- Figure 3-11. (a) Client-to-server binding using a daemon.

End point, general design issues

Figure 3-11. (b) Client-to-server binding using a superserver.

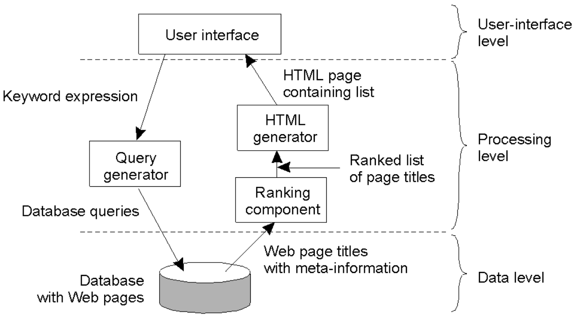


(b)

Application layering

- The user-interface level
- The processing level
- The data level

Application layering

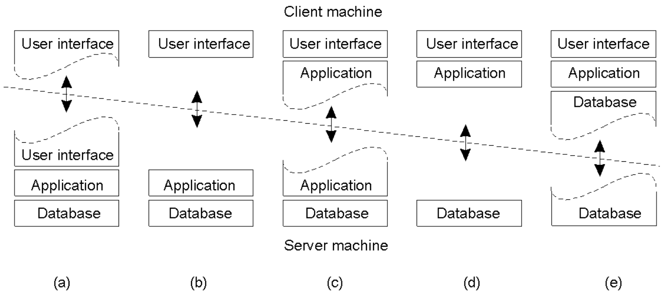


The general organization of an Internet search engine into three different layers

Component distribution (vertical)

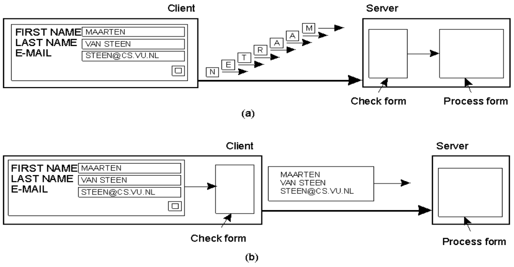
- Could have variations on component distribution
- Different amount of functionality between client-server
  - Only UI at client
  - UI+partial processing at client
  - UI+processing at client, data at server

Physical two-tired architectures



Alternative client-server organizations (a) – (e).

Server offloading



The difference between letting:

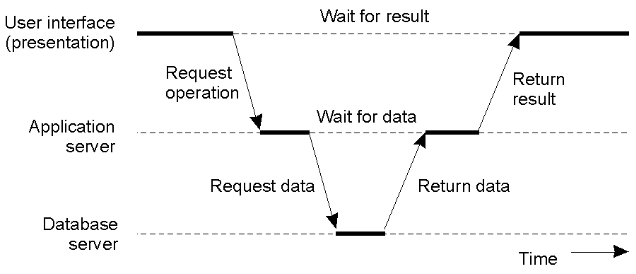
a) a server or

b) a client check forms as they are being filled

Multi-tiered servers

- Server is not necessarily a single machine
- Multi-tiered architecture:
  - Front-end
  - Application server
  - Database
- Vertical distribution

Multi-tiered architectures

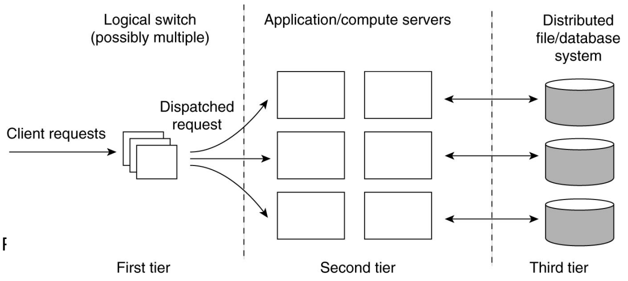


An example of a server acting as a client.

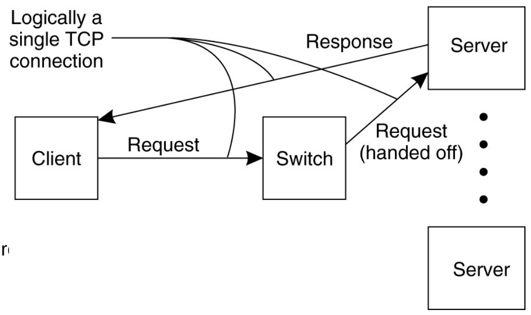
Server clusters

- Replication of functionality across machines
  - Multiple front-ends, app servers, databases
- Client requests are distributed among the servers
  - Load balancing
  - Content-aware forwarding
- Horizontal distribution

Server clusters

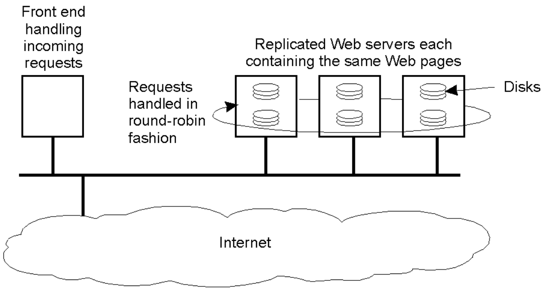


Server clusters



Figur

Modern Architectures



An example of horizontal distribution of a Web service.

Replica selection

- Round robin
- Load-based policies
- Payload-based methods (e.g., priorities)
- Energy/resource usage aware policies (e.g., costs)
- …

Replicating state

- Non-trivial problem
- Challenges
  - Ensuring replica consistency
  - Avoding too heavy performance penalties
  - Fault management
- Requires proper notions of order and state
  - Distributed algorithms
  - More on this in part C

Hierarchical architectures

- Tree of nodes
- Centralized architecture between parent and children
- More scalable than a centralized architecture
  - Each node handles only part of the network

Peer-to-peer systems

- All nodes are equal
- How to organise structurally?
- How to find other nodes?

Overlay networks

- A logical network consisting of participant components (processes/machines)
- Built on top of physical network
- Can be thought of as a graph
- Nodes are processes/machines, links are communication channels (e.g., TCP connections)

Types of peer-to-peer systems

- **Unstructured:** Built in a random manner
  - Each node can end up with any sets of neighbors, any part of application data
  - E.g.: Gnutella, Kazaa
- **Structured:** Built in a deterministic manner
  - Each node has well-defined set of neighbors, handles specific part of application data
  - E.g.: CAN, Chord, Pastry

Hybrid architectures

- Combination of peer-to-peer and client-server
  - Some parts of the system organized as client-servers
  - Other parts organized as peer-to-peer networks

Content distribution networks (CDNs)

- Provide localized content to users
- Decentralized set of content servers, may have P2P relationship
- Client-Server relation to the users
- E.g., Akamai

Collaborative distributed systems

- Work by user collaboration
- P2P in functionality
- Startup is done in a client-server manner
- E.g., Bittorrent, Napster

Other service model variations

- Multiple servers and caches (proxies)
- Mobile code
- Mobile agents
- Low-cost computers at client side (networked computers, and thin clients)
- Mobile devices
- …