System Models – Distributed Architecture

Adapted from From

DistributedSystems: Concepts and Design, Coulouris, Dollimore and Kindberg
Edition 4, © Addison-Wesley 2005

Distributed System Models

- **Architectural Models** – Placement of parts in a distributed system and the relationship between them.
- **Fundamental Models** – Description of properties that are present in all distributed architectures.
  - **Interaction Models** – Issues dealing with the interaction of process such as performance and timing of events.
  - **Failure Models** – Specification of faults that can be exhibited by processes and communication channels.
  - **Security Models** – Threats to processes and communication channels

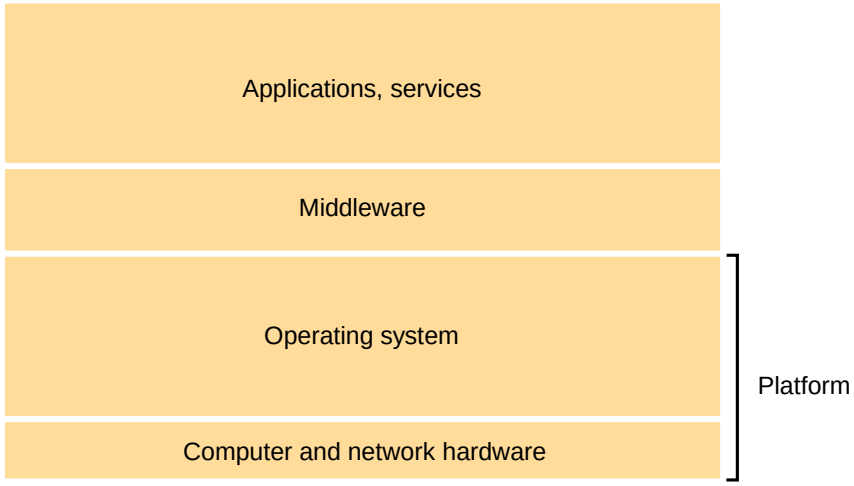
Distributed System Challenges

- Varying modes of use:
  - Load, connectivity, conflicting requirements
- Wide range of system environments:
  - Heterogeneous HW, OS, networks, scale
- Internal challenges:
  - Non-synchronized clocks, data updates, many modes of HW & SW failure modes
- External threats:
  - Attacks on data integrity & secrecy, denial of service

Architecture Models

- Distributed Architectures:
  - Placement of components across a network for the useful distribution of data and workload
  - Define functional roles of components and patterns of communication between them
- Driven by non-functional requirements:
  - Performance, reliability, security, cost, etc.

Software and hardware service layers in distributed systems



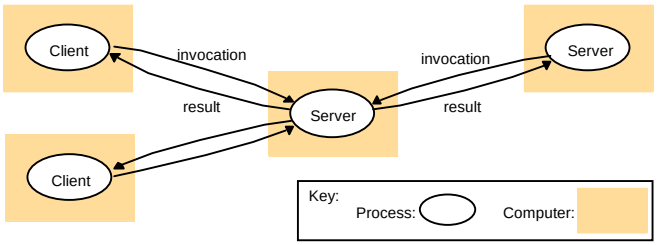
Software and hardware service layers in distributed systems

- Platform :
  - Box dependent HW & SW layers that provide services to OS layers above
- Middleware:
  - Layer of software that masks heterogeneity and provides a convenient programming model for application programmers.
  - Java RMI, CORBA, web services,DCOM(.NET Remoting)
  - Services – transactions, persistence, naming, etc.

Limitations of Middleware

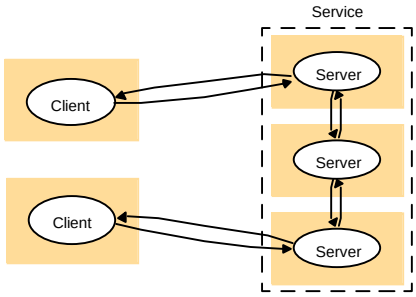
- Simplifies much distributed development, but still requires support at application level
- "end-to-end argument"
  - Saltzer, Reed & Clark (1984)
  - All communication activities cannot be completely abstracted away
  - Affects design decisions

Client–Server Model



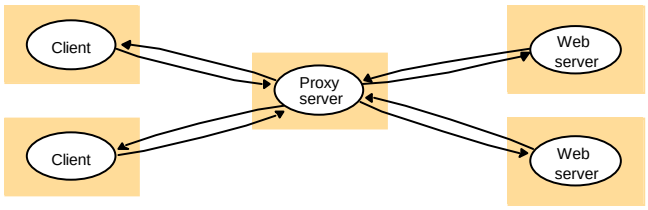
- Historically the most important and most widely used
- Servers may in turn be clients of other servers
- Simple approach to sharing data and resources, but scales poorly (bounded by capacity of server host & bandwidth of network)

Variation -a service provided by multiple servers



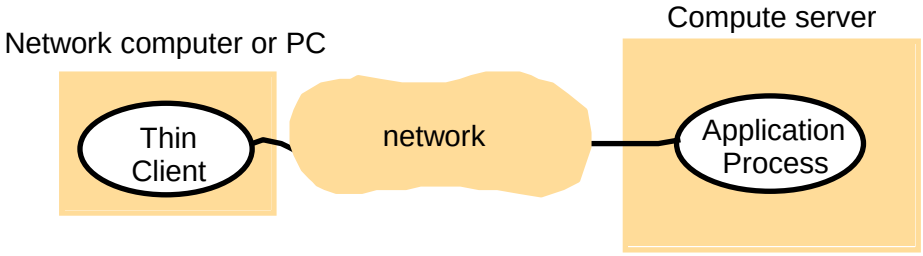
- Partition set of objects, or replicate copies on several hosts
- Web as an example

Variation - Web proxy server



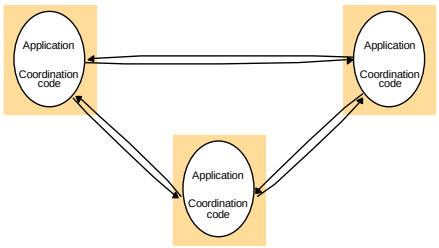
- Supports concept of a cache – store of recently used data objects
- Web browser cache
- Web proxy servers – shared cache of resources for multiple clients
- Increases availability and performance by reducing load on network and web servers.

Variation - Thin clients and compute servers



- Thin client – supports a window based interface only
- Compute server – cluster or multiprocessor computer running the application
- Classic example - X-11 Window System
- Highly interactive graphical applications tax the network and compute server

A distributed application based on peer processes



- Processes interact cooperatively as peers – no client/server distinction
- Data and resources are collectively stored and managed
- Addresses client/server boundary of computing and communication load
- Objects replicated to distribute load and increase availability
- Classic example - Napster

Design Requirements for Distributed Architectures

- Performance Issues
  - Response Time – time system takes to process a request.
  - Responsiveness – how quickly system acknowledges a request (as opposed to processing it)
  - Throughput – rate at which work gets done
  - Latency – time needed to accomplish remote requests
  - Note that "performance" can be viewed from conflicting perspectives (throughput vs response time)

Guidelines for Designing for Performance

"Make it work, make it right, make it fast" – Kent Beck

- Start with a "good enough" design (make it work)
- Make the system meet functional requirements (make it right)
- Take measurements to accurately access performance – refactor as needed (make it fast)
- Don't forget maintainability issues – tradeoff readability/understandability against performance.

http://c2.com/cgi/wiki?MakeItWorkMakeItRightMakeItFast