Distributed Systems (CS 543)

Introduction

Dongman Lee

Dept of CS

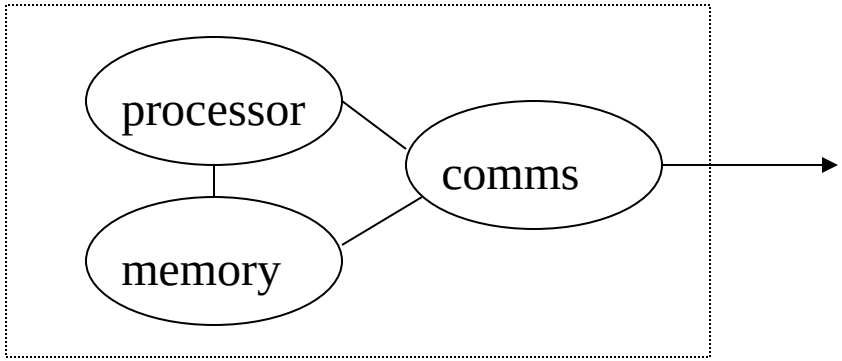
KAIST

Class Overview

- Definition and Motivation of Distributed Systems
- Properties of Distributed Systems
- Challenges of Distributed Systems
- Design Principles of Distributed Systems
- Distributed System Architecture and Model

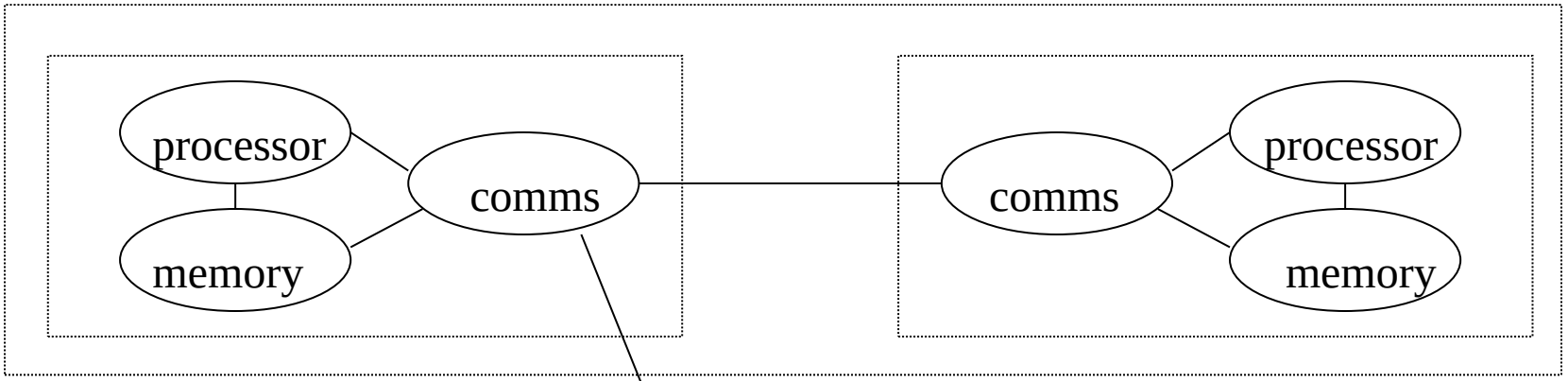
Introduction

- A system is:
  - an autonomous whole
  - an owner of a set of resources
  - something that can perform information processing
  - something that may be able to communicate with other systems



Introduction (cont.)

- A distributed system
  - a system
  - is composed of other systems
  - requires explicit communication among the components
  - its components have a common goal set
  - ➔ Asynchronous system
    - ➔ Concurrency
    - ➔ No global clock
    - ➔ Independent failures



Definitions

- A distributed system is one in which hardware or software components at networked computers communicate and coordinate their actions only by passing messages [CDK]
- A distributed system is a collection of independent computers that appears to its users as a single coherent system [Tanenbaum]
- Consists of a collection of autonomous computers linked by a computer network and equipped with distributed system software [CDK]
- Several computers doing something together; multiple components, interconnections and shared state [Schroeder]
- The fundamental properties are fault tolerance and parallelism [Mullender]
- A distributed system is one that stops you from getting any work done when a machine you've never heard of crashes [Lamport]

Why Build Distributed Systems?

- inherently distributed
  - people, information, etc.
- performance/cost
  - users requires more CPU cycles: e.g. interactive UI
  - network bandwidth growth >> CPU clock speed
  - concurrency
- modularity
  - standard interfaces
  - CORBA, WWW, and Internet are examples
- expandability
  - incremental growth

Why Build Distributed Systems? (cont.)

- availability
  - partial failure
  - load sharing
- scalability
  - ideally no limit
  - no qualitative change as the system scales
  - careful design required to scale to very large numbers of components: e.g. naming, addressing, etc.
- reliability
  - fail-safe
  - recovery: roll-back

Examples of Distributed Systems

- Internet
  - Heterogeneity
  - Scalability
  - Fault tolerance
- Mobile and ubiquitous computing
  - Location & hand-off management
  - Service discovery & integration management
  - Spontaneous operation support
  - Security
- Cloud computing
  - Security
  - Availability
  - Resource sharing

Properties of Distributed Systems

- Fundamental property of distributed systems
  - ➔ **separation**
- Derived properties
  - Isolation property
  - Explicit communication property
  - Location property
  - Heterogeneity property
  - Multiple authority property
  - Concurrency property
  - Incremental change property
  - Partial failure property

Properties of Distributed Systems (cont.)

- Isolation property
  - explicit access and potential for control over accessibility of components
- Explicit communications property
  - components have disjoint storage: explicit communications between components
  - communication mechanism between components
- Location/mobility property
  - components are potentially separable
  - requires explicit communications
  - possibility of relocation - location change

Properties of Distributed Systems (cont.)

- Heterogeneity property
  - implication of diverse implementation technology usage
  - mechanism to accommodate heterogeneous components
- Multiple authority property
  - implication of multiple autonomous management or control authorities
  - mechanism to make systems consistent
  - multiple security domains
- Concurrency property
  - timely parallel activity to occur
  - synchronization
  - modeling of relative and temporal event ordering

Properties of Distributed Systems (cont.)

- Incremental change property (extensibility)
  - potential to add or remove components
- Partial failure property
  - potential to continue after failure of individual property
  - failure recovery and fault modeling

Challenges of Distributed Systems

- Heterogeneity
- Openness
- Scalability
- Concurrency
- Security

Challenges of Distributed Systems (cont.)

- Heterogeneity
  - Applies to
    - Networks
      - interfaces, protocols
    - Hardware
      - data representation
    - Operating systems
      - system calls
    - Programming languages
      - data representation
  - Support for heterogeneity
    - Middleware
      - DCE, CORBA, DCOM
    - Virtual machine/mobile code
      - Java

Challenges of Distributed Systems (cont.)

- Openness
  - Criteria to determine the extensibility or re-configurability of a system
    - Be able to interact with services from other components, regardless of heterogeneity of the underlying environment
  - Key factors to openness: coherence
    - public interfaces to key functions
    - uniform communication mechanism and public interfaces for access to shared resources
    - conformance test of components to integrate
  - Implementing openness
    - provide only mechanisms, not policies
    - examples
      - level of consistency for client-cached data
      - operations for mobile code download
      - network QoS requirement adjustment
      - level of security

Challenges of Distributed Systems (cont.)

- Scalability
  - Said to be *scalable* if a system
    - remains effective in the presence of a significant increase in the number of resources and users [CJK]
    - can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity [Neuman]
  - Scalability components [Neuman]
    - size scalability
    - geographic scalability
    - administrative scalability
  - Design considerations for scalability
    - cost of physical resources
    - performance loss
    - software resource shortage
    - performance bottleneck

Challenges of Distributed Systems (cont.)

- Scalability (cont.)
  - Techniques for scaling
    - distribution
      - distribute data and computations across multiple sites
    - replication
      - replicate data to multiple sites
    - caching
      - make copies available locally
  - Scalability trade-offs
    - consistency vs. global synchronization

Challenges of Distributed Systems (cont.)

- Failure handling
  - A system is considered *faulty* once its behavior is no longer consistent with its specification [Schneider]
    - *partial failure property*
  - Failure handling techniques
    - Fault detection:
      - omission failure, timing failure (performance failure), response failure, crash failure
    - Fault masking
      - retransmission, checksum, roll-back
    - Fault tolerance
      - can detect a fault and either fail predictably or mask the fault from users
    - Recovery from failures
      - roll-back
    - Redundancy
      - k-resilience

Challenges of Distributed Systems (cont.)

- Concurrency
  - Concurrent access to a shared resource may cause inconsistency of the resource
  - Inconsistency examples
    - lost updates
      - two transactions concurrently perform an update operation
    - inconsistent retrievals
      - performing retrieval operation before or during an update operation
  - To avoid possible problems due to concurrent access, operations of related transactions must be *serialized (one-at-a-time)*

Challenges of Distributed Systems (cont.)

- Security
  - Authentication
    - verification of source
  - Authorization
    - access right to the resource
  - Encryption and decryption
    - public vs. private key

Considerations in Distributed System Design

- Widely varying modes of use
  - Workload
  - Connectivity
  - Timeliness
- Wide range of system environments
  - Heterogeneity
  - Performance
  - Scalability
- Internal problems
  - Synchronization
  - Failure
- External threats
  - Security

Distributed Systems Design Principles [Mullender]

- Replicate to increase availability
  - replication and consistency vs. availability
- Tradeoff availability and consistency
  - network name service vs. bank transaction
- Cache hints if possible
  - vital technique for high-performance distributed system design and implementation
- Stashing to allow autonomous operation
  - conceal the temporal disconnection from networks
- Exploit locality with caches
  - cache coherence protocols

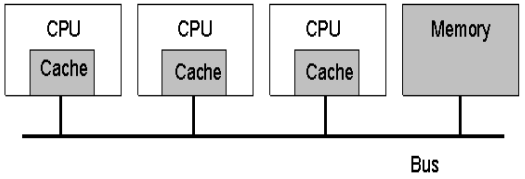
Distributed Systems Design Principles (cont.)

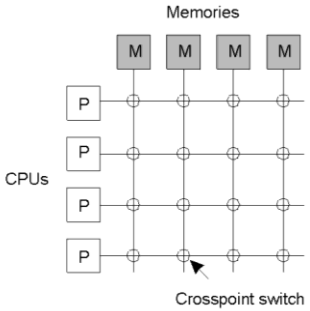
- Use timeout for revocation
  - resource locking, validity of cache, etc.
- Use a standard remote invocation mechanism
- Trust only programs on physically secure machines
- Use encryption for authentication and data security
- Try to prove distributed algorithms
  - formal specification of operations
- Capabilities might be useful
  - authentication and access right embedded in the client's request

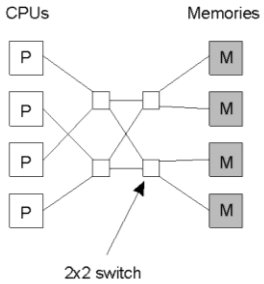
Distributed System Architecture: HW

- Multi-processors: bus-based vs. switch-based

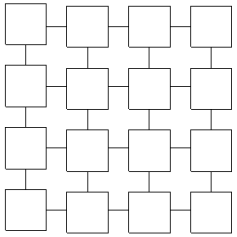
  => not a distributed system

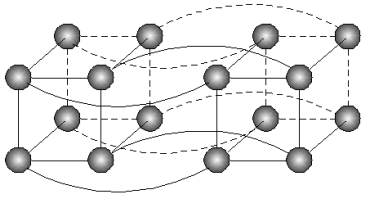






- Multi-computers
  - homogeneous systems
    - bus-based vs. switch-based
  - heterogeneous systems
    - node heterogeneity
    - network heterogeneity





→distributed systems hide heterogeneity

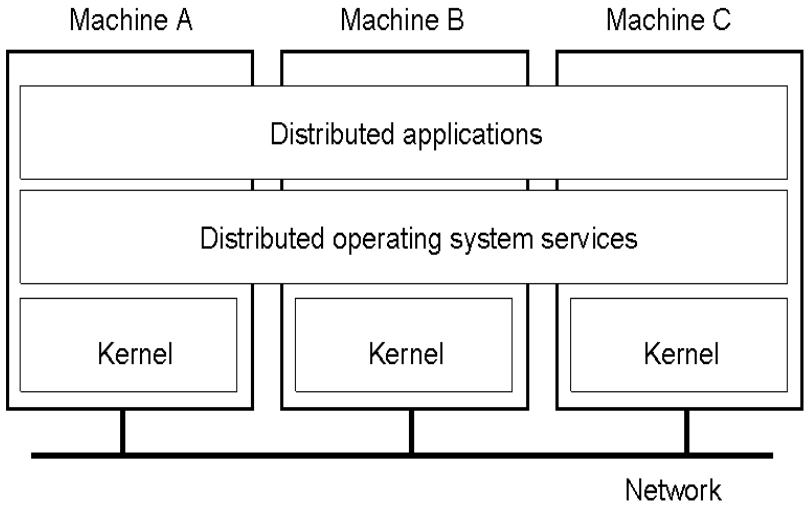
Distributed System Architecture: SW

- Distributed operating systems
- Network operating systems
- Middleware

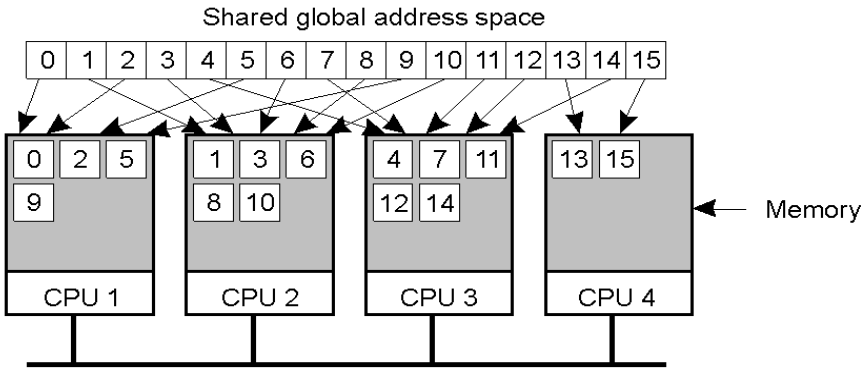
System	Description	Main Goal
DOS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency

Distributed System Architecture: SW (cont.)

- Distributed operating systems: a single system view
  - Multicomputer OS
  - Distributed shared memory system



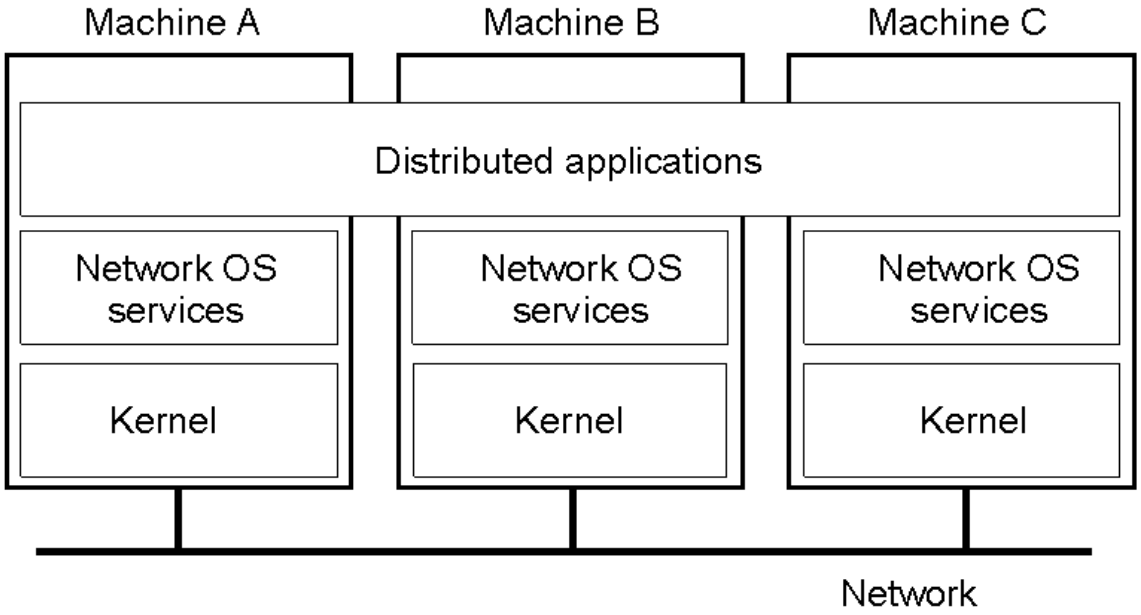
Multicomputer OS



Distributed Shared Memory

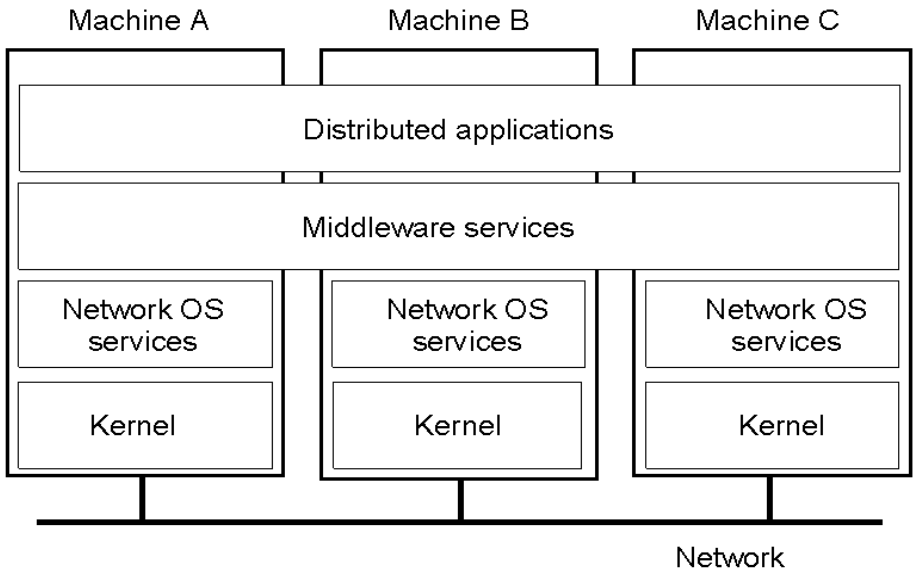
Distributed System Architecture: SW (cont.)

- Network operating systems
  - transparency via network OS services



Distributed System Architecture (cont.)

- Middleware
  - Harmonization of pros of DOS and NOS
    - DOS: transparency & ease of use
    - NOS: scalability & openness
  - Improve distribution transparency of NOS



Distributed System Architecture: SW (cont.)

- Comparison

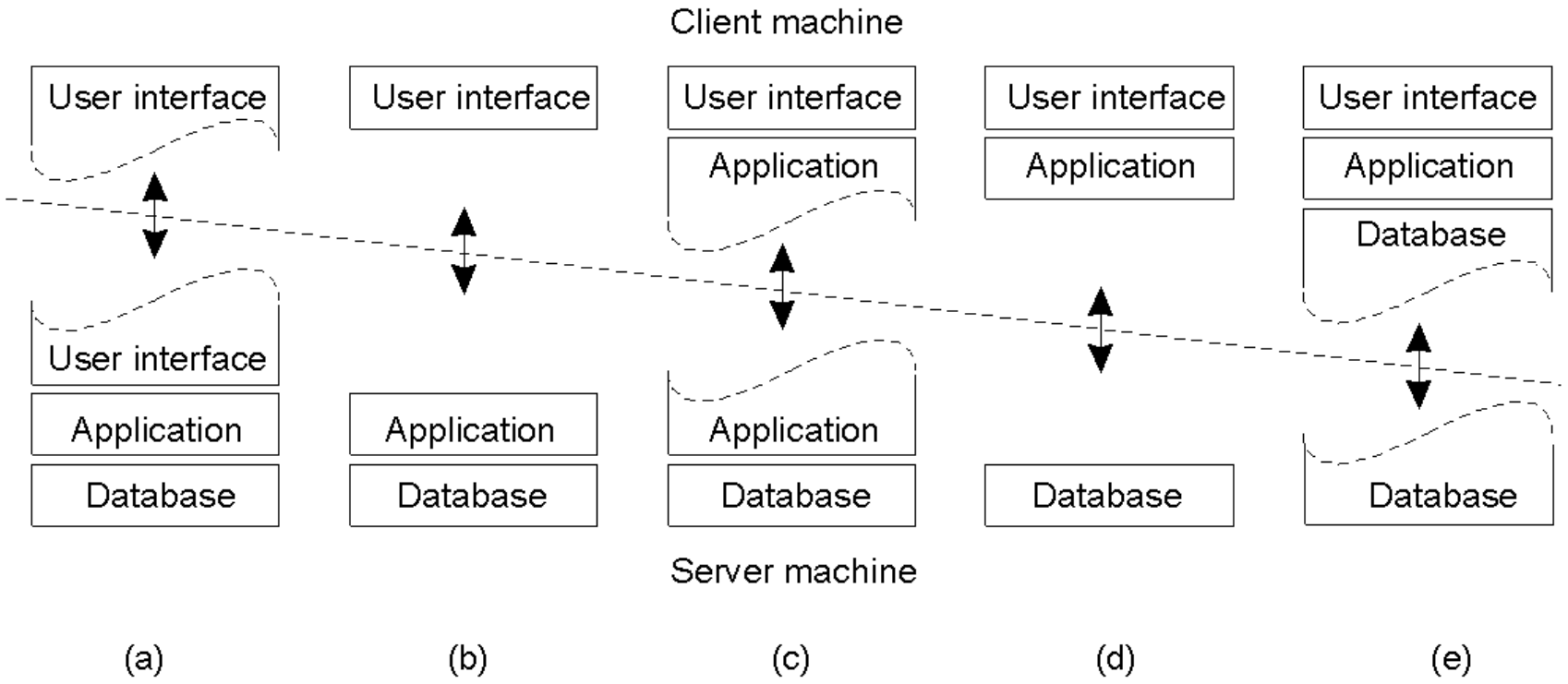
Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open

Distributed System Architecture:SW (cont.)

- Client-server model
  - Client : a process wishing to access the resources on a different computer
  - Server : a process managing the shared resources which is allowed to a client
    - HTTP server vs. Web browser
- Peer-to-peer model
  - Processes without any distinction between clients and servers
  - Distribution of control and load

Distributed System Architecture: SW (cont.)

- Multi-tier architecture
  - Alternatives

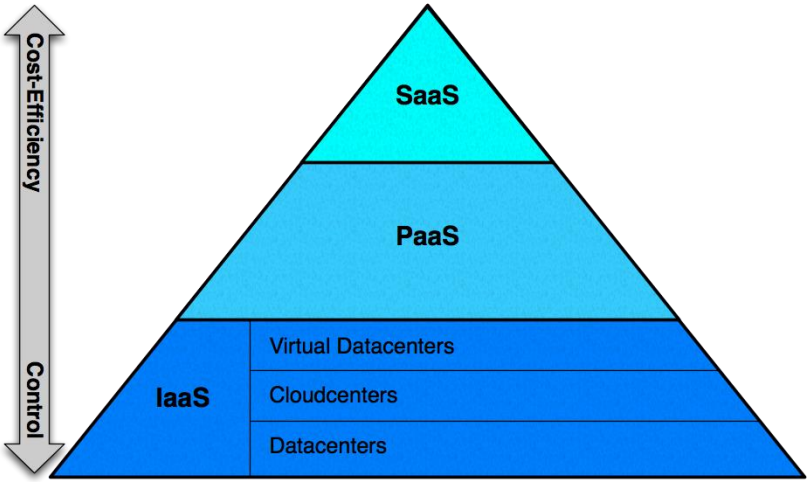


Variations

- Services provided by multiple servers
  - Functional distribution
  - Replication
- Proxy servers and caches
  - Increase availability and performance of the service
- Mobile code
  - Code at the client downloaded to the server
    - e.g., Applets
- Mobile agents
  - Executing program (code and data) that goes from one computer to another in a network carrying out a task
    - e.g., worm program (Xerox PARC)

Variations (cont.)

- Cloud computing
  - IaaS | Infrastructure as a Service: server, storage, and network resources
  - PaaS | Platform as a Service : application sw and service development platform
  - SaaS | Service as a Service : application sw



Variations (cont.)

- Ubiquitous computing
  - the form of distribution that integrates mobile devices and other devices into a given network
  - Key features
    - easy connection to a local network
      - A device brought into a new network environments is transparently reconfigured to obtain connectivity there
    - easy integration with local services
      - automatic discovery of available services with no special configuration
      - discovery services
  - Issues
    - supporting convenient connection and integration
    - limited connectivity
      - how the system can support the user so that they can continue to work while disconnected
    - security and privacy
      - track of users' location

Variations (cont.)

- Ubiquitous computing (cont.)
  - Discovery services
    - to accept and store details of services that are available on the network and to respond to queries from clients about them
    - Interfaces of discovery services
      - registration service : accept registration requests from servers, stores properties in database of currently available services
      - lookup service : match requested services with available servers
  - Context acquisition and inference
  - Dynamic reconfiguration

Variations (Cont.)

- Ubiquitous computing – Distributed IoT
  - Strategy & structure for effective data acquisition & formalization
  - Context-aware data mining
  - QoS-driven service coordination and execution
  - Semantic networking
  - Security and Trust
  - Self-*