Distributed Computing Systems

Overview of Distributed Systems

Andrew Tanenbaum and Marten van Steen, *Distributed Systems – Principles and Paradigms*, Prentice Hall, c2002.

Outline

- Overview
- Goals
- Software
- Client Server

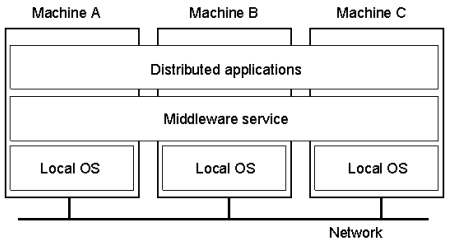
The Rise of Distributed Systems

- Computer hardware prices falling, power increasing
  - If cars did same, Rolls Royce would cost 1 dollar and get 1 billion miles per gallon (with 200 page manual to open door)
- Network connectivity increasing
  - Everyone is connected with "fat" pipes, even when moving
- It is *easy* to connect hardware together
  - Layered abstractions have worked very well

- Definition: a *distributed system* is

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

Depiction of a Distributed System



Examples:
- The Web
- Processor pool
- Shared memory pool
- Airline reservation
- Network game

- Distributed system organized as middleware. Note that middleware layer extends over multiple machines.
- Users can interact with system in consistent way, regardless of where interaction takes place (e.g., RPC, `memcached`, …
- Note: Middleware may be "part" of application in practice

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 a resource may be copied
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 Problems

- As systems grow, centralized solutions are limited
  - Consider LAN name resolution (ARP) vs. WAN

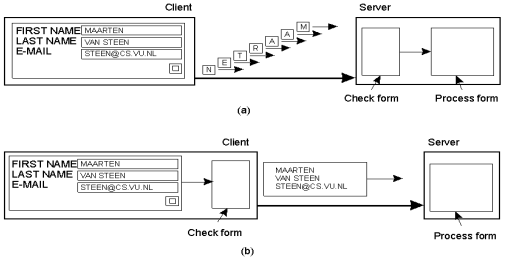
Concept	Example
Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized algorithms	Doing routing based on complete information

- Ideally, can collect information in distributed fashion and distribute in distributed fashion
- But sometimes, hard to avoid (e.g., consider money in a bank)
- Challenges: geography, ownership domains, time synchronization
- Scaling techniques? → Hiding latency, distribution, replication (next)

Scaling Technique: Hiding Communication Latency

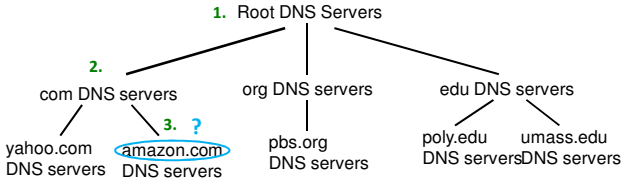
- Especially important for interactive applications
- If possible, do *asynchronous communication* – continue working so user does not notice delay

  - Not always possible when client has nothing to do
- Instead, can hide latencies



Scaling Technique: Distribution

- Spread information/processing to more than one location



Client wants IP for www.amazon.com (*approximation*):

1. Client queries root server to find `.com` DNS server
2. Client queries `.com` DNS server to get `amazon.com` DNS server
3. Client queries `amazon.com` DNS server to get IP address for `www.amazon.com`

Scaling Technique: Replication

- Copy of information to increase availability and decrease centralized load
  - Example: File caching is replication decision made by client
  - Example: CDNs (e.g., Akamai) for Web
  - Example: P2P networks (e.g., BitTorrent) distribute copies uniformly or in proportion to use
- Issue: Consistency of replicated information
  - Example: Web browser cache or NFS cache – how to tell it is out of date?

Outline

- Overview (done)
- Goals (done)
- Software (next)
- Client Server

Software Concepts

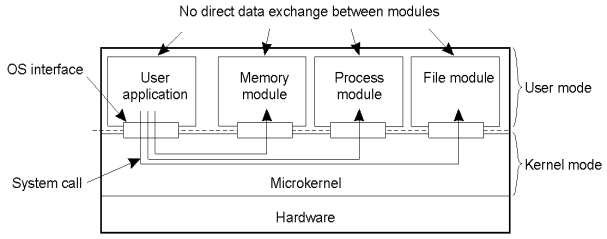
System	Description	Main Goal
DOS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer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

- DOS (Distributed Operating Systems)
- NOS (Network Operating Systems)
- Middleware

(Next)

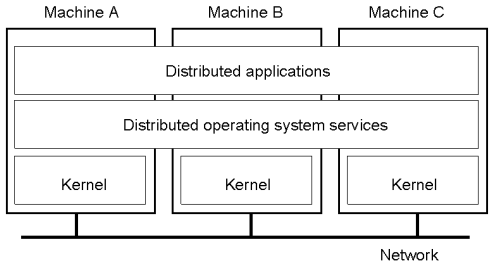
Distributing Single-Computer Operating Systems

- Separating applications from operating system code with *microkernel*



Can extend to multiple computers (see next slide)

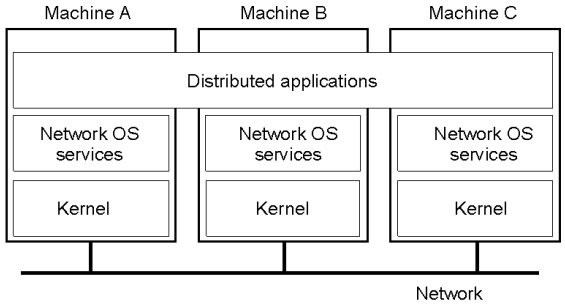
Distributed Operating Systems



- Typically, all hosts are homogenous
- But no longer have shared memory
  - Can try to provide *distributed shared memory*
    - But tough to get acceptable performance, especially for large requests

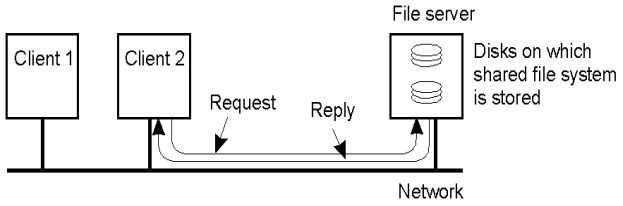
  → Provide *message passing*

Network Operating System



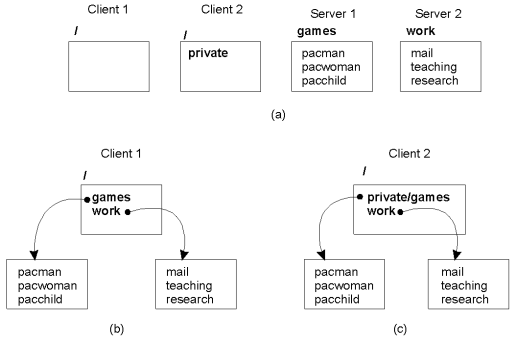
- OSes can be different (Windows or Linux)
- Typical services: `rlogin`, `rcp`
  - Fairly primitive way to share files

Network Operating System



- Can have one computer provide files transparently for others (NFS)

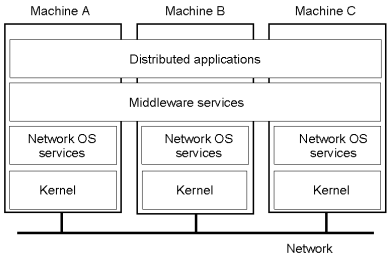
Network Operating System



- Different clients may mount the servers in different places
- Inconsistencies in view make NOSes harder for users than DOSes
  - But easier to scale by adding computers

Positioning Middleware

- Network OS not transparent. Distributed OS not independent of computers.
  - Middleware can help



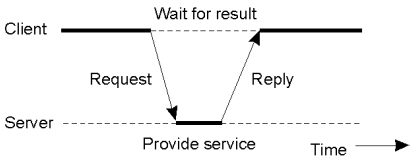
• Often middleware built in-house to help use networked operating systems (distributed transactions, better comm, RPC)
  — Unfortunately, many different standards

Outline

- Overview (done)
- Goals (done)
- Software (done)
- Client Server (next)

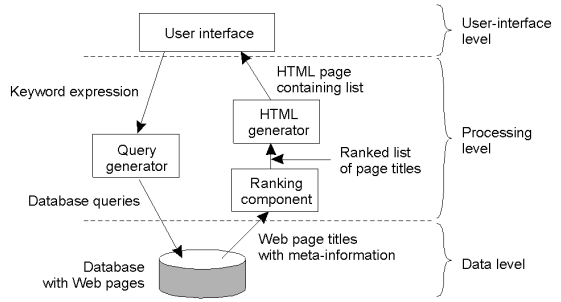
Clients and Servers

- Thus far, have not talked about organization of processes
  - Again, many choices but most widely used is *client-server*



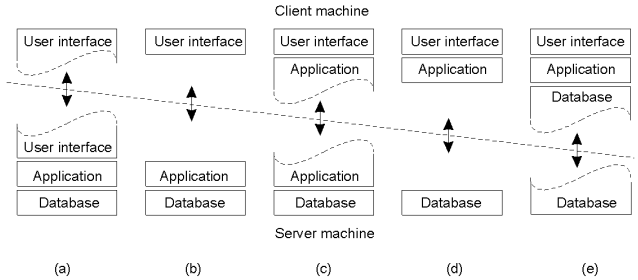
- If can do so without connection, quite simple
  - If underlying connection is unreliable, not trivial
  - Resend. What if receive twice?
- Use TCP for reliable connection (most Internet apps)
  - Not always appropriate for high-speed LAN connection or interactive applications

Client-Server Implementation Levels



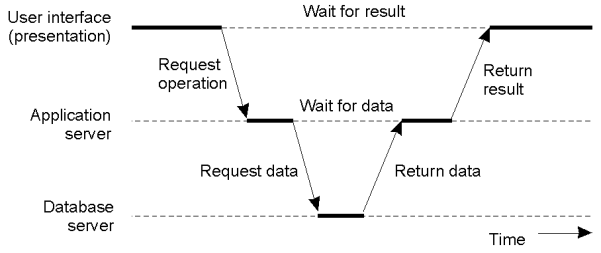
- Example of Internet search engine
  - UI on client
  - Data level is server, keeps consistency
  - Processing can be on client *or* server

Multitiered Architectures



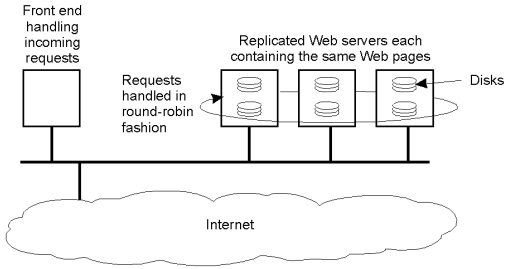
- Thin client (a) to Fat client (e)
  - (a) is simple echo terminal, (b) has GUI at client
  - (c) has user side processing (e.g., check Web form for consistency)
  - (d) and (e) popular for NOS environments (e.g., server has files only)

Multitiered Architectures: 3 tiers



- Server(s) may act as client(s), sometimes
  - Example: transaction monitor across multiple databases
- Also known as *vertical distribution*

Alternate Architectures: Horizontal



- Rather than vertical, distribute servers across nodes
  - Example: Web server "farm" for load balancing
  - Clients, too (peer-to-peer systems)
  - Most effective for read-heavy systems