CIS 505: Software Systems

Introduction to Distributed Systems

Insup Lee
Department of Computer and Information Science
University of Pennsylvania

CIS 505, Spring 2007

Distributed Systems

- Why distributed systems?
  - availability of powerful yet cheap microprocessors (PCs, workstations, PDAs, embedded systems, etc.)
  - continuing advances in communication technology
- What is a distributed system?
  - A distributed system is a collection of independent computers that appear to the users of the system as a single coherent system.

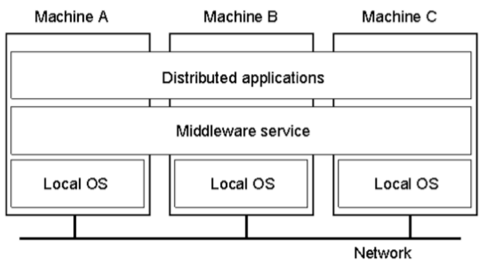
Examples

- The world wide web – information, resource sharing
- Clusters, Network of workstations
- Distributed manufacturing system (e.g., automated assembly line)
- Network of branch office computers - Information system to handle automatic processing of orders
- Network of embedded systems
- New Cell processor (PlayStation 3)

Advantages and disadvantages

- Advantages
  - Economics
  - Speed
  - Inherent distribution
  - Reliability
  - Incremental growth
- Disadvantages
  - Software
  - Network
  - More components to fail
  - Security

Organization of a Distributed System



A distributed system organized as middleware.
Note that the middleware layer extends over multiple machines.

Goals of Distributed Systems

- Transparency
- Openness
- Reliability
- Performance
- Scalability

Transparency

- How to achieve the single-system image, i.e., how to make a collection of computers appear as a single computer.
- Hiding all the distribution from the users as well as the application programs can be achieved at two levels:
  1) hide the distribution from users
  2) at a lower level, make the system look transparent to programs.

  1) and 2) requires uniform interfaces such as access to files, communication.

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 shared by several competitive users
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.

Openness

- Make it easier to build and change
- Monolithic Kernel: systems calls are trapped and executed by the kernel. All system calls are served by the kernel, e.g., UNIX.
- Microkernel: provides minimal services.
  - IPC
  - some memory management
  - some low-level process management and scheduling
  - low-level i/o (E.g., Mach can support multiple file systems, multiple system interfaces.)
- Standard interface, separation of policy from mechanism

Reliability

- Distributed system should be more reliable than single system.
  - Availability: fraction of time the system is usable. Redundancy improves it.
  - Need to maintain consistency
  - Need to be secure
  - Fault tolerance: need to mask failures, recover from errors.
- Example: 3 machines with .95 probability of being up
  - (1-.05)**3 vs 1-.05**3 probability of being up

Performance

- Without gain on this, why bother with distributed systems.
- Performance loss due to communication delays:
  - fine-grain parallelism: high degree of interaction
  - coarse-grain parallelism
- Performance loss due to making the system fault tolerant.

Scalability

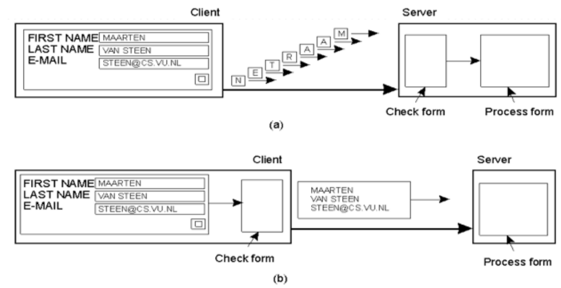
- Systems grow with time or become obsolete.
- Techniques that require resources linearly in terms of the size of the system are not scalable. (e.g., broadcast based query won't work for large distributed systems.)
- Examples of bottlenecks (i.e., scalability limitations)
  - Centralized components: a single mail server
  - Centralized tables/data: a single URL address book
  - Centralized algorithms: routing based on complete information

Scalability Problems

Characteristics of decentralized algorithms:

- No machine has complete information about the system state.
- Machines make decisions based only on local information.
- Failure of one machine does not ruin the algorithm.
- There is no implicit assumption that a global clock exists.

Scaling Techniques (1)

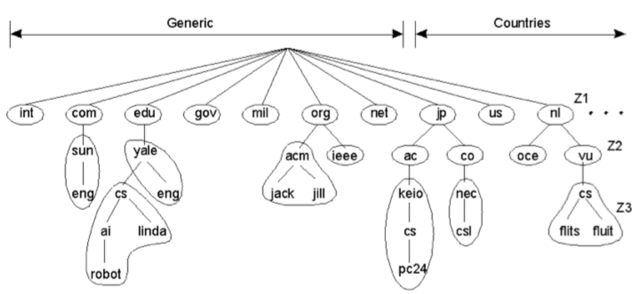


The difference between letting:

a) a server or

b) a client check forms as they are being filled

Scaling Techniques (2)



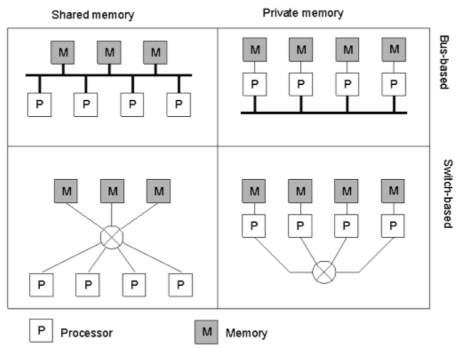
An example of dividing the DNS name space into zones.

Pitfalls when Developing Distributed Systems

False assumptions made by first time developer:

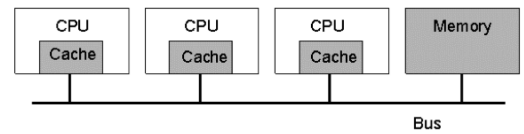
- The network is reliable.
- The network is secure.
- The network is homogeneous.
- The topology does not change.
- Latency is zero.
- Bandwidth is infinite.
- Transport cost is zero.
- There is one administrator.

Hardware Concepts



Different basic organizations and memories in distributed computer systems: multiprocessors vs. multicomputers

Multiprocessors (1)

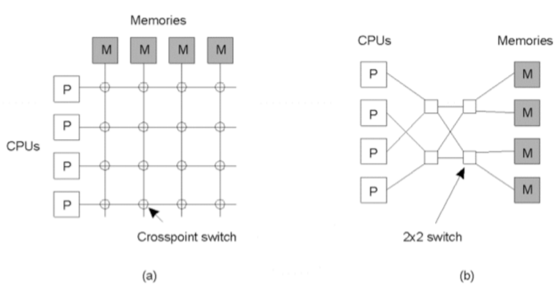


A bus-based multiprocessor

- Cache memory, hit rate, coherence, write-through cache, snoopy cache

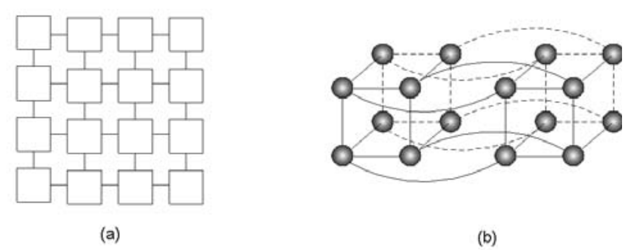
Multiprocessors (2)

a) A crossbar switch
b) An omega switching network



Homogeneous Multicomputer Systems

a) Grid
b) Hypercube



Tightly coupled vs. loosely coupled

How slow is the network?

- “ping www.cis.upenn.edu”
- Round-trip times
  - Upenn .5ms
  - Princeton 5ms
  - Rice 43ms
  - Stanford 80ms
  - Tsinghua, Beijing 280ms

Communication Latency

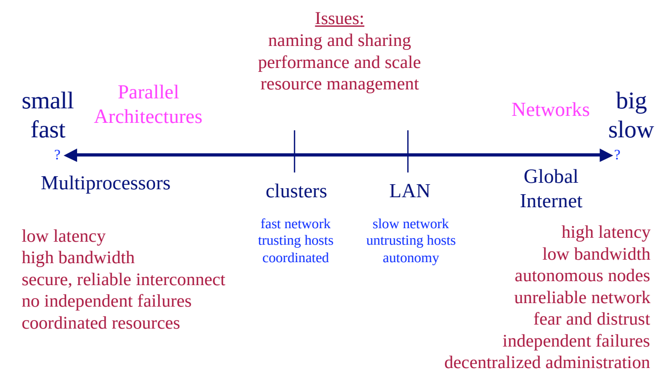
- Latency – “wire delay”
  - Time to send and recv one byte of data
  - Depends on “distance”
- Bandwidth
  - Bytes/second
  - Depends on size of vehicle
- Latency is the bottleneck
  - It improves slower than bandwidth
    - Speed of light
    - Routers in the middle (traffic stops)
  - Request-respond cycles dominate application

The speed pyramid

register	1
L2	10
Memory	200
LAN	100,000
Disk	2,000,000
WAN	20,000,000

- Will the ratios change?

Continuum of Distributed Systems

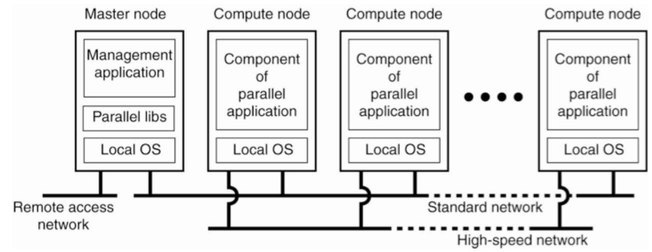


[J. Chase]

Types of Distributed Systems

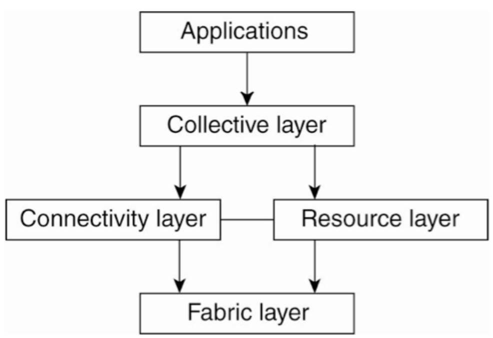
- Distributed Computing Systems
- Distributed information systems
- Distributed Pervasive/Embedded Systems

Cluster Computing Systems



- Figure 1-6. An example of a cluster computing system.

Grid Computing Systems



- Figure 1-7. A layered architecture for grid computing systems.

Transaction Processing Systems (1)

Primitive	Description
BEGIN_TRANSACTION	Mark the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

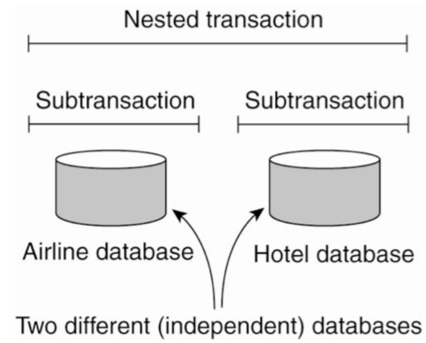
- Figure 1-8. Example primitives for transactions.

Transaction Processing Systems (2)

Characteristic properties of transactions:

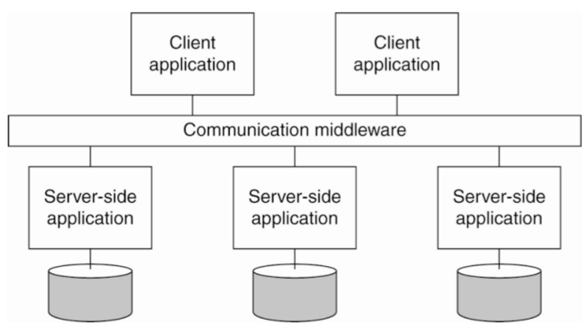
- Atomic: To the outside world, the transaction happens indivisibly.
- Consistent: The transaction does not violate system invariants.
- Isolated: Concurrent transactions do not interfere with each other.
- Durable: Once a transaction commits, the changes are permanent.

Transaction Processing Systems (3)



- Figure 1-9. A nested transaction.

Enterprise Application Integration



- Figure 1-11. Middleware as a communication facilitator in enterprise application integration.

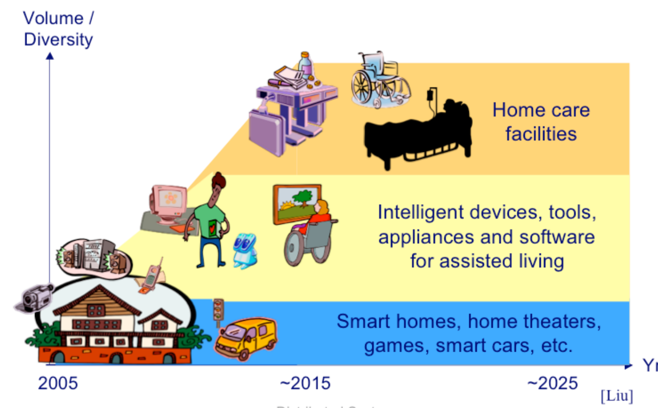
Distributed Pervasive Systems

Requirements for pervasive systems

- Embrace contextual changes.
- Encourage ad hoc composition.
- Recognize sharing as the default.

Embedded Home Environment

Example: Home and Personal Appliances



Justifications

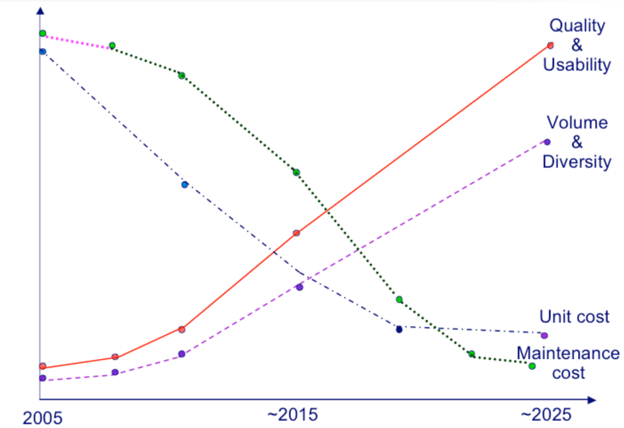
- Rapid advances in component technologies, e.g.,
  - o Smart gadgets, wearable sensors and actuators, robotic helpers, mobile devices
  - o Wireless, wideband interconnects
- Increasing critical needs due to
  - o Aging baby-boom generation
  - o Long life expectancy
  - o New safety, security, and privacy concerns

Observations

- Number of users: 10 – 1000 million
- Types of sensors and actuators: 100's
- Number of suppliers: 10 – 100's
- Required reliability: <10,000 recalls/year
- User tolerance to glitches: minimum
- Product life cycles: 3 – 20 yrs
- Tolerable upgrade effort: minimum

The environment must be open and evolvable, & capable of self diagnosis, healing, maintenance

Desired Trends

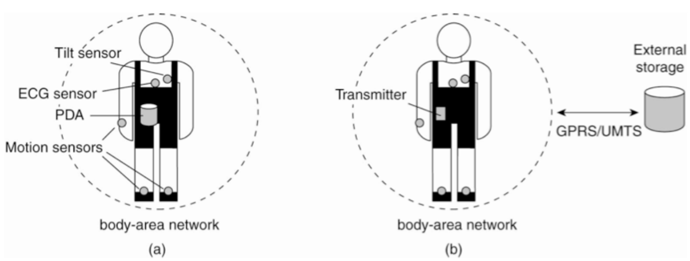


Electronic Health Care Systems (1)

Questions to be addressed for health care systems:

- Where and how should monitored data be stored?
- How can we prevent loss of crucial data?
- What infrastructure is needed to generate and propagate alerts?
- How can physicians provide online feedback?
- How can extreme robustness of the monitoring system be realized?
- What are the security issues and how can the proper policies be enforced?

Electronic Health Care Systems (2)



- Figure 1-12. Monitoring a person in a pervasive electronic health care system, using (a) a local hub or
- (b) a continuous wireless connection.

Background: Sensor Networks

- Types of sensors
  - o Seismic, low sampling rate magnetic, thermal, visual, infrared, acoustic and radar
- Conditions to monitor
  - o Temperature, humidity, (vehicular) movement, lightning condition, pressure, soil makeup, noise levels
  - o Presence or absence of certain kinds of objects
  - o Mechanical stress levels on attached objects
  - o Current characteristics such as speed, direction, and size of an object

Technology Trends: Mote Evolution

Mote Type Year	*WeC* 1998	*René* 1999	*René 2* 2000	*Dot* 2000	*Mica* 2001	*Mica2Dot* 2002	*Mica 2* 2002	*Telos* 2004
								
Microcontroller								
Type	AT90LS8535		ATmega163		ATmega128			TI MSP430
Program memory (KB)	8		16		128			48
RAM (KB)	0.5		1		4			10
Active Power (mW)	15		15		15		60	0.5
Sleep Power (μW)	45		45		75		75	2
Wakeup Time (μs)	1000		36		180		180	6
Nonvolatile storage								
Chip	24LC256				AT45DB041B			ST M24M01S
Connection type	I^2C				SPI			I^2C
Size (KB)	32				512			128
Communication								
Radio	TR1000				TR1000	CC1000		CC2420
Data rate (kbps)	10				40	38.4		250
Modulation type	OOK				ASK	FSK		O-QPSK
Receive Power (mW)	9				12	29		38
Transmit Power at 0dBm (mW)	36				36	42		35
Power Consumption								
Minimum Operation (V)	2.7		2.7		2.7			1.8
Total Active Power (mW)	24				27	44	89	38.5
Programming and Sensor Interface								
Expansion	none	51-pin	51-pin	none	51-pin	19-pin	51-pin	10-pin
Communication	IEEE 1284 (programming) and RS232 (requires additional hardware)							USB
Integrated Sensors	no	no	no	yes	no	no	no	yes

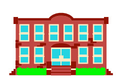
SN Characteristics

- Environment
  - connect to physical environment (large numbers, dense, real-time)
  - Sensor nodes are prone to failures, non-deterministic
  - wireless communication
  - massively parallel interfaces (to users and applications)
  - Limited resources: battery, bandwidth, memory, CPU (power management critical)
- Network
  - Topology changes dynamically
  - sporadic connectivity
  - new resources entering/leaving
  - large amounts of redundancy
  - self-configure/re-configure

SN applications

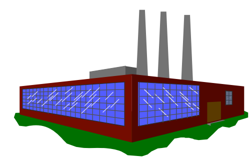
- Infrastructure security, military applications
- Environmental and Habitat monitoring
- Health applications
- Smart space/home applications
- Other commercial applications
  - Industrial Sensing
  - Traffic Control, vehicle tracking and detection
  - Interactive museums

Smart Spaces



Smart School

Smart City



Smart Factory

Other Applications

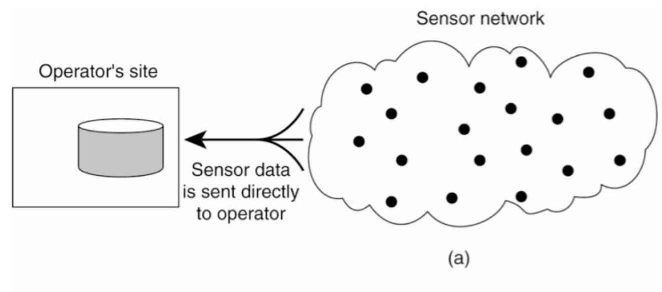
- Battlefields/Surveillance
- Earthquake areas
- Environmental Monitoring
- Airport security
- Emergency Response
- Location Services

Sensor Networks (1)

Questions concerning sensor networks:

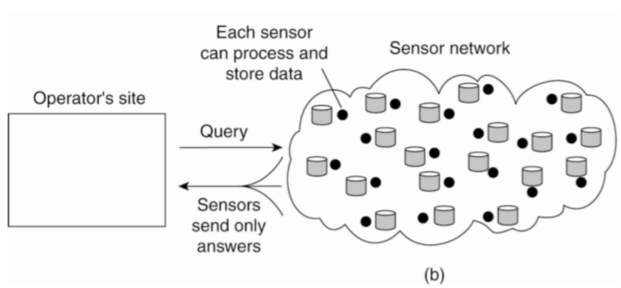
- How do we (dynamically) set up an efficient tree in a sensor network?
- How does aggregation of results take place? Can it be controlled?
- What happens when network links fail?

Sensor Networks (2)



- Figure 1-13. Organizing a sensor network database, while storing and processing data (a) only at the operator's site or …

Sensor Networks (3)



- Figure 1-13. Organizing a sensor network database, while storing and processing data … or (b) only at the sensors.

The Challenges of Distributed Systems

- Secure communication over public networks
  - ACI: who sent it, did anyone see it, did anyone change it
- Fault-tolerance
  - Building reliable systems from unreliable components
  - nodes fail independently; a distributed system can "partly fail"
  - [Lamport]: "A distributed system is one in which the failure of a machine I've never heard of can prevent me from doing my work."
- Replication, caching, naming
  - Placing data and computation for effective resource sharing, hiding latency, and finding it again once you put it somewhere.
- Coordination and shared state
  - What should the system components do and when should they do it?
  - Once they've all done it, can they all agree on what they did and when?