Introduction to Distributed Systems

Module A1

Distributed & Cloud Computing

Sheheryar Malik, Ph.D.

1

What is a Distributed Systems

A distributed system is:

A collection of autonomous computing elements that appears to its users as a single coherent system

Autonomous computing elements, also referred to as nodes, be they hardware devices or software processes
Single coherent system: users or applications perceive a single system ⇒ nodes need to collaborate

2

Distributed System

- A distributed system could be;
  - A single system with multiple processors
  - A cluster of workstation
- It is a collection of heterogeneous computers
- Resources resides in separate units
- Communication is done through message passing or shared memory
- A standardized distributed system architecture removes complexity resulting from diversity of technology
- The performance of a distributed system is determined by the speed and latency to end-to-end communication

Collection of Autonomous Nodes

- Independent behavior
  - Each node is autonomous and will thus have its own notion of time: there is no global clock. Leads to fundamental synchronization and coordination problems.
- Collection of nodes
  - How to manage group membership?
  - How to know that you are indeed communicating with an authorized (non)member?

Organization

- Overlay network
  - Each node in the collection communicates only with other nodes in the system, its neighbors
  - The set of neighbors may be dynamic, or may even be known only implicitly (i.e., requires a lookup)
- Overlay types

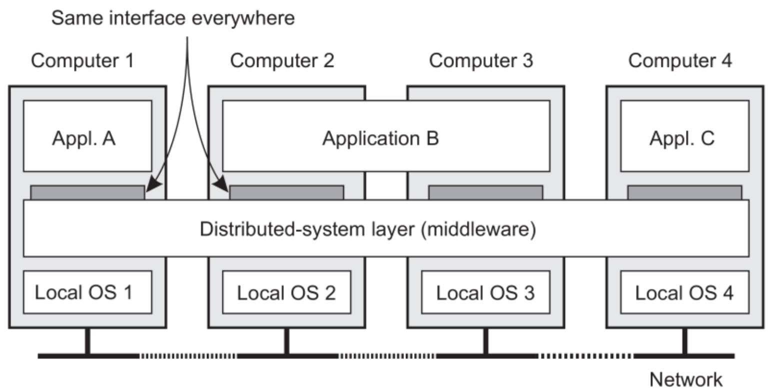
  Well-known example of overlay networks: peer-to-peer systems
  - Structured: each node has a well-defined set of neighbors with whom it can communicate (tree, ring)
  - Unstructured: each node has references to randomly selected other nodes from the system

Coherent system

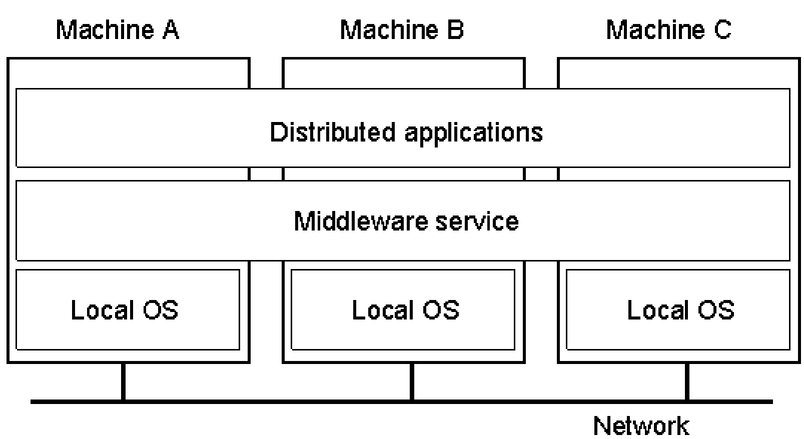
- Essence
  - The collection of nodes as a whole operates the same, no matter where, when, and how interaction between a user and the system takes place
- Examples
  - An end user cannot tell where a computation is taking place
  - Where data is exactly stored should be irrelevant to an application
  - If or not data has been replicated is completely hidden
- Keyword is distribution transparency
- The snag: partial failures
  - It is inevitable that at any time only a part of the distributed system fails
  - Hiding partial failures and their recovery is often very difficult and in general impossible to hide

Middleware: the OS of distributed systems

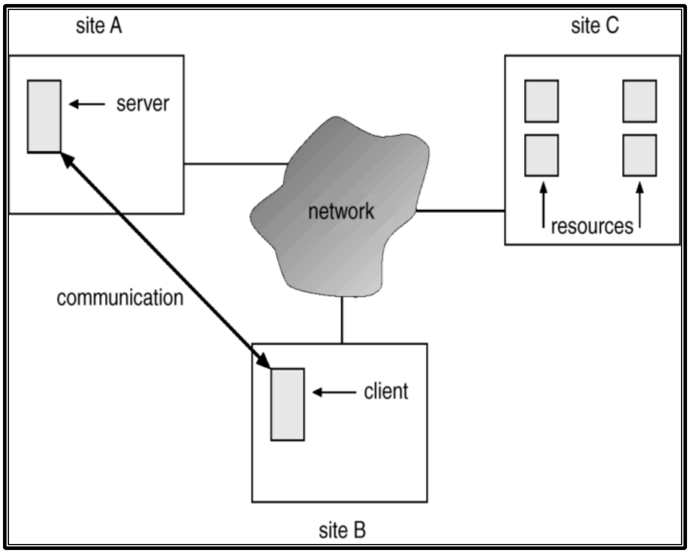
- What does it contain?
  - Commonly used components and functions that need not be implemented by applications separately



Distributed System



Distributed System



Advantages of Distributed System

- Communication and resource sharing possible
  - Data sharing
    - one computer can obtain access to data held at some other computer
  - Function sharing
    - it enables one computer to use facilities available on some other computer
- Economical : Price - performance ratio
  - Enable one application to utilized several computer resources
- Reliability
  - The effect of breakdown of one or more components can be reduced
- Scalability
  - It has potential for incremental growth

Distributed OS vs Network OS

Network Operating System

- Users are aware of multiplicity of machines
- Access to resources of various machines is done explicitly by:
  - Remote logging into the appropriate remote machine
  - Transferring data from remote machines to local machines, via the File Transfer Protocol (FTP) mechanism

Distributed Operating System

- Users not aware of multiplicity of machines
  - access to remote resources similar to access to local resources
- Data Migration
  - transfer data by transferring entire file, or
  - transferring only those portions of the file necessary for the immediate task
- Computation Migration
  - transfer the computation, rather than the data, across the system

Distributed Application Examples

- The World Wide Web
- Automated banking systems
- Cellular phone network
- Global positioning systems
- Retail point-of-sale terminals
- Air-traffic control
- Avionics (fly-by-wire)

Motivation for Distribution

Resource sharing

Personalise environments

Location independence

People & information are distributed

Performance & cost

Modularity & expandability

Availability & reliability

Scalability

Sharing Resources

- Canonical examples
  - Cloud-based shared storage and files
  - Peer-to-peer assisted multimedia streaming
  - Shared mail services (think of outsourced mail systems)
  - Shared Web hosting (think of content distribution networks)

Design Goals & Challenges

Heterogeneity	Openness	Scalability
Security	Failure handling	Concurrency
Performance	Consistency	Transparencies

15

Heterogeneity

- Variety and differences in
  - Computer hardware
  - Operating systems
  - Networks
  - Programming languages
  - Implementations by different developers
- Middleware (a software layer) is used to deal with heterogeneity
  - It provide a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware, OS, and programming languages (e.g., CORBA)

16

Openness

- Openness determines how the system can be extended and implemented in various ways
- Openness is achieved by publishing the key interfaces
- Openness is concerned with extensions and improvements of distributed systems
- New components have to be integrated with existing components
- Differences in data representation of interface types on different processors (of different vendors) have to be resolved

Openness of Distributed Systems

- Open distributed system
  - Be able to interact with services from other open systems, irrespective of the underlying environment
    - Systems should conform to well-defined interfaces
    - Systems should easily interoperate
    - Systems should support portability of applications
    - Systems should be easily extensible
- Achieving openness
  - At least make the distributed system independent from heterogeneity of the underlying environment
    - Hardware
    - Platforms
    - Languages

Implementing Openness
Policies versus Mechanisms

- Implementing openness: policies
  - What level of consistency do we require for client-cached data?
  - Which operations do we allow downloaded code to perform?
  - Which QoS requirements do we adjust in the face of varying bandwidth?
  - What level of secrecy do we require for communication?
- Implementing openness: mechanisms
  - Allow (dynamic) setting of caching policies
  - Support different levels of trust for mobile code
  - Provide adjustable QoS parameters per data stream
  - Offer different encryption algorithms

Scalability of Distributed Systems

- Scalability is adaptation of distributed systems to
  - accommodate more users
  - respond faster (this is the hard one)
- It is done to
  - control the performance loss
  - prevent software resources running out
  - avoid performance bottleneck
- Usually done by adding more and/or faster processors
- Components should not need to be changed when scale of a system increases
- Design components to be scalable

Scalability of Distributed Systems

- At least three components
- Most system easily scale with size
  - Actual challenge resides with geographical and administrative scalability

Size scalability

Number of users and/or processes

Geographical scalability

Maximum distance between nodes

Administrative scalability

Number of administrative domains

Salability Problems in Size Scalability

- Root causes for scalability problems with centralized solutions
  - The computational capacity, limited by the CPUs
  - The storage capacity, including the transfer rate between CPUs and disks
  - The network between the user and the centralized service

Salability Problems in Geographical Scalability

- Cannot simply go from LAN to WAN
  - many distributed systems assume synchronous client-server interactions: client sends request and waits for an answer
  - Latency may easily prohibit this scheme
- WAN links are often inherently unreliable
  - simply moving streaming video from LAN to WAN is bound to fail
- Lack of multipoint communication, so that a simple search broadcast cannot be deployed
  - Solution is to develop separate naming and directory services (having their own scalability problems)

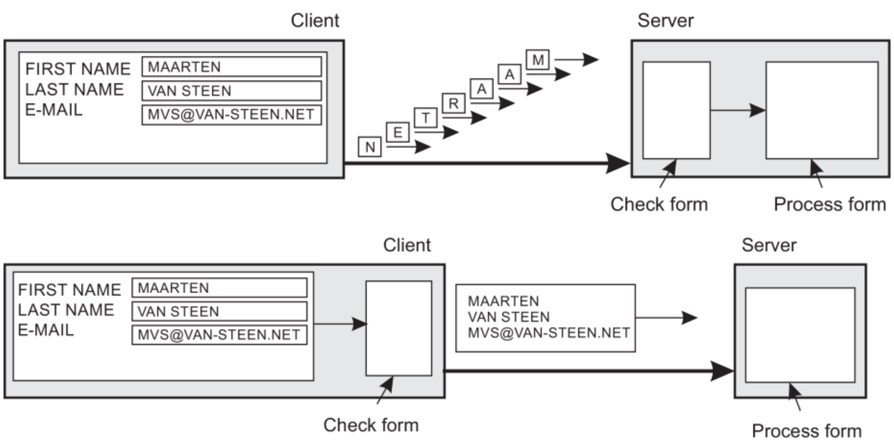
Salability Problems in Administrative Scalability

- Conflicting policies concerning usage (and thus payment), management, and security
- Examples
  - Computational grids: share expensive resources between different domains
  - Shared equipment: how to control, manage, and use a shared radio telescope constructed as large-scale shared sensor network?
- Exception: several peer-to-peer networks
  - File-sharing systems (e.g., BitTorrent)
  - Peer-to-peer telephony (Skype)
  - Peer-assisted audio streaming (Spotify)

Techniques for Scalability in Distributed System

- Hide communication latencies
  - Avoid waiting for responses; do something else
    - Make use of asynchronous communication
    - Have separate handler for incoming response
    - Problem: not every application fits this model
- Partition & Distribution
  - Partition data and computations across multiple machines
    - Move computations to clients (Java applets)
    - Decentralized naming services (DNS)
    - Decentralized information systems (WWW)
- Replication/caching
  - Make copies of data available at different machines
    - Replicated file servers and databases
    - Mirrored Web sites
    - Web caches (in browsers and proxies)
    - File caching (at server and client)

Techniques for Scalability in Distributed System



Facilitate solution by moving computations to client

Scalability in Distributed System
The Problem with Replication

- Applying replication is easy, except for one thing
  - Having multiple copies (cached or replicated), leads to inconsistencies: modifying one copy makes that copy different from the rest.
  - Always keeping copies consistent and in a general way requires global synchronization on each modification.
  - Global synchronization precludes large-scale solutions.

- If we can tolerate inconsistencies, we may reduce the need for global synchronization, but tolerating inconsistencies is application dependent.

Security

- Security is one the important concern while designing a distributed systems as the things are going to be shared
- In a distributed system, clients send requests to access data managed by servers, resources in the networks
  - Doctors requesting records from hospitals
  - Users purchase products through electronic commerce
- Security is required for
  - Concealing the contents of messages: security and privacy
  - Identifying a remote user or other agent correctly (authentication)
- New challenges
  - Denial of service attack
  - Security of mobile code

Failure Handling (Fault Tolerance)

- Hardware, software and networks fail
- Distributed systems must maintain availability even at low levels of hardware/software/network reliability
- Failure handling in distributed system is mainly concerned with
  - Detecting failure
  - Masking failure
- Fault tolerance is achieved by
  - recovery
  - redundancy

Concurrency

- Components in distributed systems are executed in concurrent processes
- Components access and update shared resources (e.g. variables, databases, device drivers)
  - Concurrency should be provided for both the process & resource side
- Integrity of the system may be violated if concurrent updates are not coordinated
  - Lost updates
  - Inconsistent analysis

Distributed System Transparencies

- Distributed systems should be perceived by users and application programmers as a whole rather than as a collection of cooperating components
- Transparency has different aspects
- These represent various properties that distributed systems should have

Distributed System Transparencies

Location transparency	Access transparency	Migration transparency
Replication transparency	Concurrency transparency	Failure transparency
Relocation transparency	Performance transparency	Scaling transparency

Distributed System Transparencies
Location Transparency

- It hides that where the resource is located
- It enables resources to be accessed without knowledge of their physical or network location
  - for example, which building or IP address
- The name of the resource does not indicate the physical location
- If users change location, their view of system would not change
- This implies the support of access transparency
- Example; Files, processors

Distributed System Transparencies
Access Transparency

- Enables local and global resources to be accessed using identical operation
- hide differences in data representation and how a resource is accessed
- enables local and remote resources to be accessed using identical operations

Distributed System Transparencies
Migration Transparency

- It hides that a resource may move to another location
- It allows the movement of resources and clients within a system without affecting the operation of users or programs
- Users cannot notice if a resource or their job has been migrated from one location to other within distributed system
- Location transparency is necessary for this to occur

35

Distributed System Transparencies
Replication Transparency

- It enables multiple instances of resources (files and servers) to be used to increase reliability and performance without knowledge of the replicas by users or application programmers
- All changes and updates must be made simultaneously to all replicas
- It increases the reliability and performance

36

Distributed System Transparencies
Concurrency Transparency

- It hides that a resource may be shared by several competing users or processes, or
- It enables several processes to operate concurrently using shared resources without interference between them
- A processor utilizes multiple resources at the same time

Distributed System Transparencies
Failure Transparency

- If a link or system in distributed system fails, the entire system should not fail
- It hides the failure and recovery of a resource or system
- It enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components

Distributed System Transparencies
Relocation Transparency

- Hide that an object may be moved to another location while in use

Distributed System Transparencies
Performance Transparency

- Allows the system to reconfigure as the load vary
- It hides the procedure and effect of load balancing and load sharing

Distributed System Transparencies
Scaling Transparency

- It hide that the new components are added to system
- It allows the system and applications to expand in scale without change to the system structure or the application algorithms

Degree of Transparency

Aiming at full distribution transparency may be too much

- There are communication latencies that cannot be hidden
  - Users may be located in different continents
- Completely hiding failures of networks and nodes is (theoretically and practically) impossible
  - You cannot distinguish a slow computer from a failing one
  - You can never be sure that a server actually performed an operation before a crash
- Full transparency will cost performance, exposing distribution of the system
  - Keeping replicas exactly up-to-date with the master takes time
  - Immediately flushing write operations to disk for fault tolerance

DS Information Management Solution

- A Distributed System can be managed in two manners
  - Distributed and centralized solution

DS Information Management Solution
Centralized Solution

- Place the entire decision/information in one location
- Easy to manage
- Application upgrade is quite simple
- Disadvantages
  - Become critical element
  - If fails, entire distributed system is subject to failure
  - Network traffic is increased towards centralized system

DS Information Management Solution
Distributed Solution

- It does not suffer from critical element
- If one system fails, entire DS will not fail
- Disadvantages
  - Increase traffic when involve broadcasting information
  - Difficult for several locations to maintain consistent information
  - It requires huge cooperation among participants

Developing Distributed Systems: Pitfalls

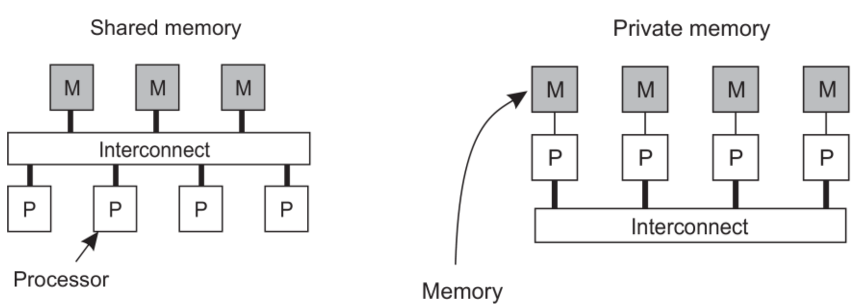
- Many distributed systems are needlessly complex caused by mistakes that required patching later on
  - Many false assumptions are often made.
- False (and often hidden) assumptions
  - The network is reliable
  - The network is secure
  - The network is homogeneous
  - The topology does not change
  - Latency is zero
  - Bandwidth is infinite
  - Transport cost is zero
  - There is one administrator

Three Types of Distributed Systems

- High performance distributed computing systems
- Distributed information systems
- Distributed systems for pervasive computing

Parallel Computing

- High-performance distributed computing started with parallel computing



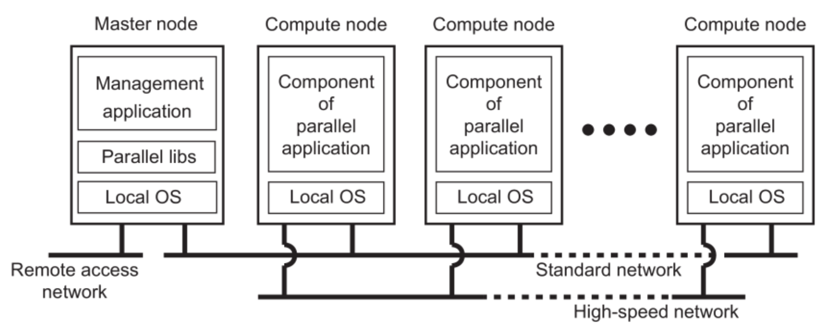
Multiprocessor and multicore versus multicomputer

Distributed Shared Memory Systems

- Multiprocessors are relatively easy to program in comparison to multicomputers, yet have problems when increasing the number of processors (or cores)
  - Solution: Try to implement a shared-memory model on top of a multicomputer
- Example through virtual-memory techniques
- Map all main-memory pages (from different processors) into one single virtual address space
  - If process at processor A addresses a page P located at processor B, the OS at A traps and fetches P from B, just as it would if P had been located on local disk
- Problem
  - Performance of distributed shared memory could never compete with that of multiprocessors, and failed to meet the expectations of programmers
  - It has been widely abandoned by now

Cluster Computing

- Essentially a group of high-end systems connected through a LAN
- Homogeneous: same OS, near-identical hardware
- Single managing node



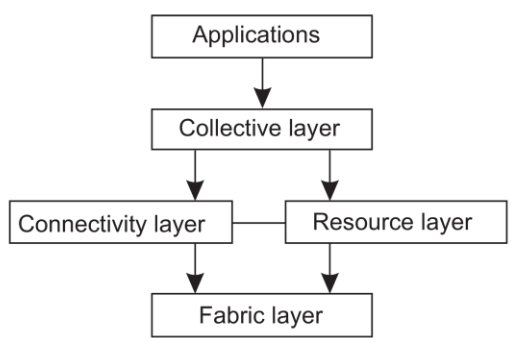
Grid Computing

- Heterogeneous
- Dispersed across several organizations
- Can easily span a wide-area network

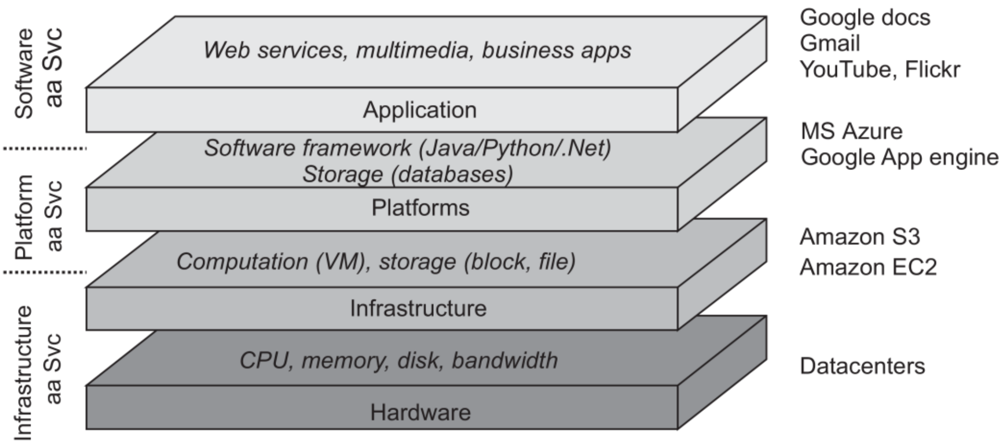
- To allow for collaborations, grids generally use virtual organizations
    - In essence, this is a grouping of users (or better: their IDs) that will allow for authorization on resource allocation

Architecture for Grid Computing: The layers

- Fabric
    - Provides interfaces to local resources (for querying state and capabilities, locking, etc.)
- Connectivity
    - Communication/transaction protocols, e.g., for moving data between resources. Also various authentication protocols
    - such as creating processes or reading data
- Collective
    - Handles access to multiple resources: discovery, scheduling, replication
- Application
    - Contains actual grid applications in a single organization



Cloud Computing



Cloud Computing Layers

- Hardware
  - Processors, routers, power and cooling systems
  - Customers normally never get to see these
- Infrastructure
  - Deploys virtualization techniques
  - Evolves around allocating and managing virtual storage devices and virtual servers
- Platform
  - Provides higher-level abstractions for storage and such
  - Example: Amazon S3 storage system offers an API for (locally created) files to be organized and stored in so-called buckets
- Application
  - Actual applications, such as office suites (text processors, spreadsheet applications, presentation applications)
  - Comparable to the suite of apps shipped with OSes

Distributed Pervasive Systems

- Emerging next-generation of distributed systems in which nodes are small, mobile, and often embedded in a larger system, characterized by the fact that the system naturally blends into the user's environment
- Three (overlapping) subtypes
  - Ubiquitous computing systems
    - pervasive and continuously present, i.e., there is a continuous interaction between system and user.
  - Mobile computing systems
    - pervasive, but emphasis is on the fact that devices are inherently mobile.
  - Sensor (and actuator) networks
    - pervasive, with emphasis on the actual (collaborative) sensing and actuation of the environment

Ubiquitous Systems: Core Elements

- Distribution
  - Devices are networked, distributed, and accessible in a transparent manner
- Interaction
  - Interaction between users and devices is highly unobtrusive
- Context awareness
  - The system is aware of a user's context in order to optimize interaction
- Autonomy
  - Devices operate autonomously without human intervention, and are thus highly self-managed
- Intelligence
  - The system as a whole can handle a wide range of dynamic actions and interactions

Mobile Computing: Distinctive Features

- A myriad of different mobile devices
  - smartphones, tablets, GPS devices, remote controls, active badges
- Mobile implies that a device's location is expected to change over time ⇒ change of local services, reachability, etc.
  - Keyword: discovery
- Communication may become more difficult
  - no stable route, but also perhaps no guaranteed connectivity ⇒ disruption-tolerant networking

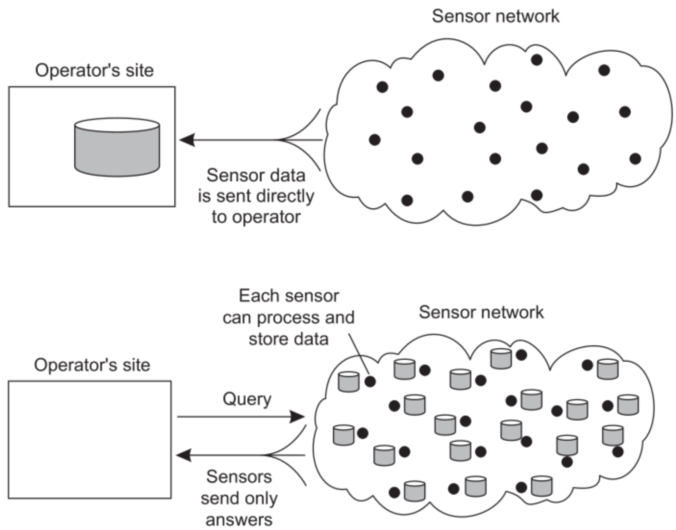
Sensor Networks

Characteristics

- The nodes to which sensors are attached are:
- Many (10s-1000s)
- Simple (small memory/compute/communication capacity)
- Often battery-powered (or even battery-less)

Sensor Networks as Distributed Databases

- Two extremes



Sensor Networks: Duty-cycled networks

- Issue
  - Many sensor networks need to operate on a strict energy budget: introduce duty cycles
- Definition
  - A node is active during T_{active} time units, and then suspended for
- $T_{suspended}$ units, to become active again. Duty cycle τ:

$$\tau = \frac{T_{active}}{T_{active} + T_{suspended}}$$

- Typical duty cycles are 10 – 30%, but can also be lower than 1%

Sensor Networks
Keeping duty-cycled networks in sync

- Issue
  - If duty cycles are low, sensor nodes may not wake up at the same time anymore and become permanently disconnected: they are active during different, nonoverlapping time slots
- Solution
  - Each node A adopts a cluster ID CA, being a number
  - Let a node send a join message during its suspended period
  - When A receives a join message from B and CA < CB , it sends a join message to its neighbors (in cluster CA) before joining B
  - When CA > CB it sends a join message to B during B's active period
- Note
  - Once a join message reaches a whole cluster, merging two clusters is very fast
  - Merging means: re-adjust clocks

Reference

- Tanenbaum, Maarten van Steen
  - Chapter 1
- Coulouris et. al.
  - Chapter 1