Distributed Systems

UNIT I Chapter 1

Lecture 1

Tannenbaum and Van Steen – Chapter 1

Definition of a Distributed System

A distributed system is a collection of independent computers that appears to its users as a single coherent system.

Important aspects:

- a distributed system consists of components (i.e., computers) that are autonomous
- users (people or programs) think they are dealing with a single system
- autonomous components need to collaborate

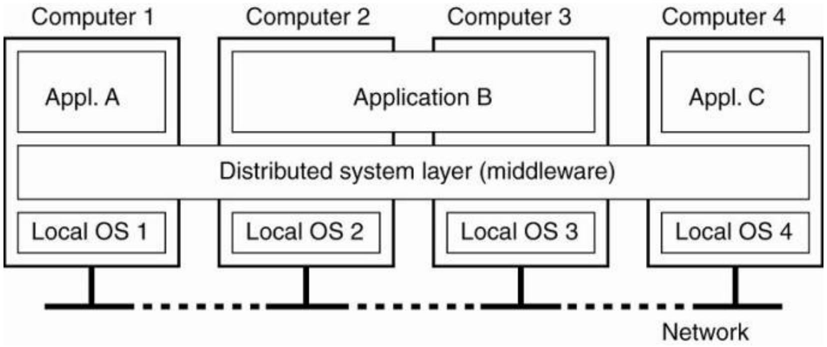
Goal of distributed systems development: establish this collaboration.

Characteristics of distributed systems:

- differences between various computers and the ways in which they communicate are mostly hidden from users
- users and applications can interact with a distributed system in a consistent and uniform way, regardless of where and when interaction takes place.
- distributed systems should also be relatively easy to expand or scale.
- a distributed system will normally be continuously available

Layers of software support heterogeneous computers and networks while offering a single-system view - sometimes called middleware.

A distributed system organized as middleware.



Four networked computers and three applications:

1. Application B is distributed across computers 2 and 3.
2. Each application is offered the same interface
3. Distributed system provides the means for components of a single distributed application to communicate with each other, but also to let different applications communicate.
4. It aso hides the differences in hardware and operating systems from each application.

Goals

Four goals that should be met to make building a distributed system worth the effort:

1. should make resources easily accessible

2. should reasonably hide the fact that resources are distributed across a network;

3. should be open

4. should be scalable.

1. Making Resources Accessible

Main goal of a distributed system –

- make it easy for the users (and applications) to access remote resources
- to share them in a controlled and efficient way.

Resources - anything: printers, computers, storage facilities, data, files, Web pages, and networks, etc.

Accessibility Issues

- security
- unwanted communication

2. Distribution Transparency

Goal - hide the fact that its processes and resources are physically distributed across multiple computers – systems should be transparent

Different forms of transparency in a distributed system (ISO, 1995).

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 is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

Degree of Transparency

Issues:

- Timing:

e.g. requesting an electronic newspaper to appear in your mailbox before 7 A.M. local time, as usual, while you are currently at the other end of the world living in a different time zone.

- Synchronization:

e.g. a wide-area distributed system that connects a process in San Francisco to a process in Amsterdam limited by laws of physics - a message sent from one process to the other takes about 35 milliseconds.

- it takes several hundreds of milliseconds using a computer network.
- signal transmission is not only limited by the speed of light, but also by limited processing capacities of the intermediate switches.

- Performance:

e.g. many Internet applications repeatedly try to contact a server before finally giving up. Consequently, attempting to mask a transient server failure before trying another one may slow down the system as a whole.

- Consistency:

e.g. need to guarantee that several replicas, located on different continents, need to be consistent all the time - a single update operation may now even take seconds to complete, something that cannot be hidden from users.

- Context Awareness:

e.g. notion of location and context awareness is becoming increasingly important, it may be best to actually expose distribution rather than trying to hide it. - consider an office worker who wants to print a file from her notebook computer. It is better to send the print job to a busy nearby printer, rather than to an idle one at corporate headquarters in a different country.

- Limits of Possibility:

Recognizing that full distribution transparency is simply impossible, we should ask ourselves whether it is even wise to pretend that we can achieve it.

3. Openness

Goal: offer services according to standard rules that describe the syntax and semantics of those services.

e.g.

- **computer networks -** standard rules govern the format, contents, and meaning of messages sent and received.

- **distributed systems** - services are specified through interfaces, which are often described in an Interface Definition Language (IDL).

➢Interface definitions written in an IDL nearly always capture only the syntax of services

- specify names of the available functions with types of parameters, return values, possible exceptions that can be raised, etc.
- allows an arbitrary process that needs a certain interface to talk to another process that provides that interface
- allows two independent parties to build completely different implementations of those interfaces, leading to two separate distributed systems that operate in exactly the same way.

Properties of specifications:

- Complete - everything that is necessary to make an implementation has been specified.
- Neutral - specifications do not prescribe what an implementation should look like

Lead to:

- Interoperability - characterizes the extent by which two implementations of systems or components from different manufacturers can co-exist and work together by merely relying on each other's services as specified by a common standard.
- Portability - characterizes to what extent an application developed for a distributed system A can be executed, without modification, on a different distributed system B that implements the same interfaces as A.

Goals: an open distributed system should also be extensible. i.e.

- be easy to configure the system out of different components (possibly from different developers).
- be easy to add new components or replace existing ones without affecting those components that stay in place.

4. Scalability

Scalability of a system is measured with respect to:

1. Size - can easily add more users and resources to the system.
2. Geographic extent - a geographically scalable system is one in which the users and resources may lie far apart.
3. Administrative scalability - can be easy to manage even if it spans many independent administrative organizations.

Scalability Limitations of Size

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

Geographical scalability Limitations

• Synchronization:

○ e.g. currently hard to scale existing distributed systems designed for local-area networks is that they are based on synchronous communication.

▪ a client requesting service blocks until a reply is sent back.

▪ works fine in LANs where communication between two machines is generally at worst a few hundred microseconds.

▪ in a wide-area system, interprocess communication may be hundreds of milliseconds, three orders of magnitude slower.

• Unreliability of communication:

○ communication in wide-area networks is inherently unreliable and point-to-point.

○ local-area networks provide reliable communication based on broadcasting, making it much easier to develop distributed systems. For example, consider the problem of locating a service.

▪ e.g. in a local-area system, a process can broadcast a message to every machine, asking if it is running the service it needs.

• Only those machines that have that service respond, each providing its network address in the reply message.

• Such a location scheme is unthinkable in a wide-area system: just imagine what would happen if we tried to locate a service this way in the Internet.

• Administrative scalability:

○ how to scale a distributed system across multiple, independent administrative domains.

▪ **major problem** - conflicting policies with respect to resource usage (and payment), management, and security.

Scaling Techniques

Three techniques for scaling:

1. hiding communication latencies

2. distribution

3. replication

Hiding communication latencies - important to achieving geographical scalability.

1. try to avoid waiting for responses to remote service requests.

• e.g, when a service has been requested at a remote machine, an alternative to waiting for a reply from the server is to do other useful work at the requester's side.

• construct the requesting application in such a way that it uses only asynchronous communication.]

2. reduce the overall communication

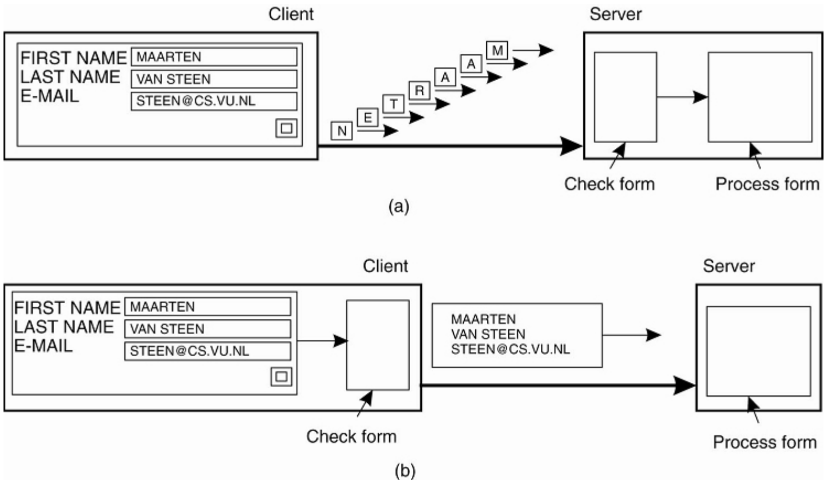
• e.g. in interactive applications when a user sends a request he will generally have nothing better to do than to wait for the answer.

• move part of the computation that is normally done at the server to the client process requesting the service.

○ typical case - accessing databases using forms.

▪ ship the code for filling in the form, and possibly checking the entries, to the client, and have the client return a completed form - approach of shipping code is now widely supported by the Web in the form of Java applets and Javascript.

The difference between letting (a) a server or (b) a client check forms as they are being filled.



(a)

(b)

Distribution - splitting a component into smaller parts and spreading those parts across the system.

• e.g. Internet Domain Name System (DNS).

○ The DNS name space is hierarchically organized into a tree of domains, which are divided into nonoverlapping zones

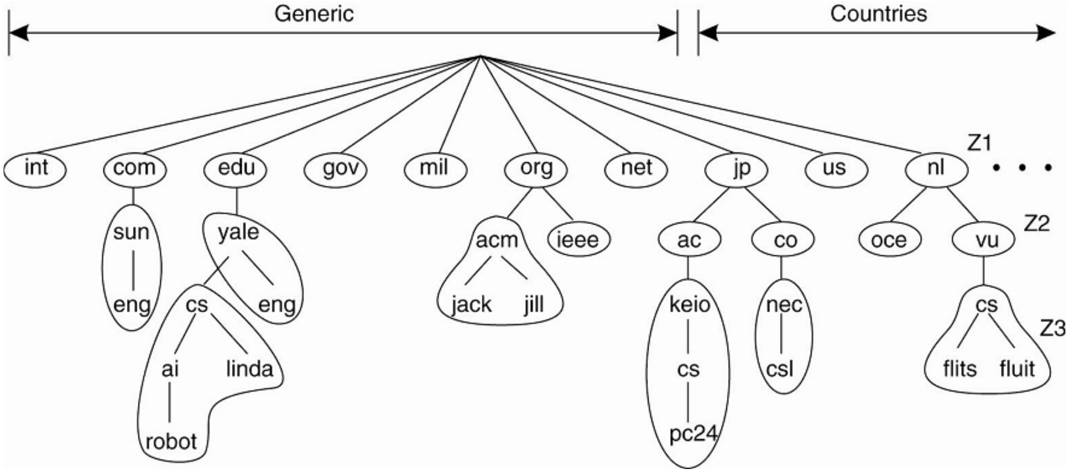
o Names in each zone are handled by a single name server.

○ Resolving a name means returning the network address of the associated host.

▪ e.g. the name nl.vu.cs.flits.

• To resolve this name - first passed to the server of zone which returns the address of the server for zone Z2, to which the rest of name, vu.cs.flits, can be handed. The server for Z2 will return the address of the server for zone Z3, which is capable of handling the last part of the name and will return the address of the associated host.

An example of dividing the DNS name space into zones.



- DNS is distributed across several machines, thus avoiding that a single server has to deal with all requests for name resolution.
- Performance degradation Problems

Solution: replicate components across a distributed system

• Replication:

o increases availability

o helps balance the load between components leading to better performance.

○ e.g. in geographically widely-dispersed systems - a copy nearby can hide much of the communication latency problems.

• Caching - special form of replication

- caching results in making a copy of a resource, generally in the proximity of the client accessing that resource.
- caching is a decision made by the client of a resource, and not by the owner of a resource.
- caching happens on demand whereas replication is often planned in advance.

Issues of caching and replication - multiple copies of a resource -> modifying one copy makes that copy different from the others -> leads to consistency problems.

- Weak consistency – e.g. a cached Web document of which the validity has not been checked for the last few minutes.
- Strong consistency – e.g. electronic stock exchanges and auctions.
- problem –

- an update must be immediately propagated to all other copies.
- if two updates happen concurrently, it is often also required that each copy is updated in the same order.
- generally requires some global synchronization mechanism – hard to implement in a scalable way (i.e. speed of light

Pitfalls

False assumptions that everyone makes when developing a distributed application for the first time (by Peter Deutsch):

1. The network is reliable.
2. The network is secure.
3. The network is homogeneous.
4. The topology does not change.
5. Latency is zero.
6. Bandwidth is infinite.
7. Transport cost is zero.
8. There is one administrator.

Types of Distributed Systems

Distributed Computing Systems

Distributed systems used for high-performance computing task

1. cluster computing - the underlying hardware consists of a collection of similar workstations or PCs, closely connected by means of a high-speed local-area network, each node runs the same operating system.

2. grid computing - constructed as a federation of computer systems, where each system may fall under a different administrative domain, and may be very different when it comes to hardware, software, and deployed network technology.

Cluster Computing Systems

- price/performance ratio of personal computers and workstations became financially and technically attractive to build a supercomputer using off-the-shelf technology by simply hooking up a collection of relatively simple computers in a high-speed network.
- In virtually all cases, cluster computing is used for parallel programming in which a single (compute intensive) program is run in parallel on multiple machines.
- example of a cluster computer - Linux-based Beowulf clusters
- Each cluster consists of a collection of compute nodes that are controlled and accessed by means of a single master node.
- master node:

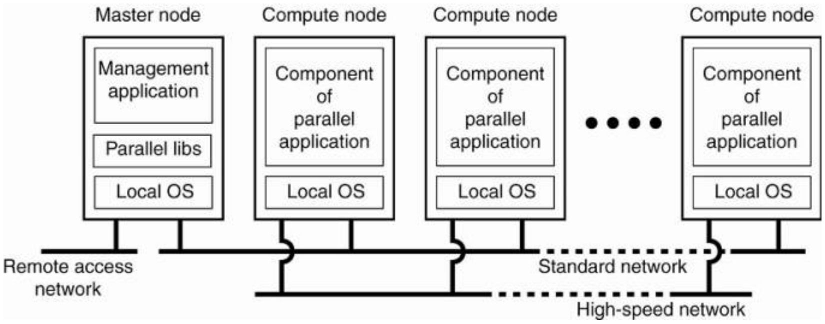
- handles the allocation of nodes to a particular parallel program
- maintains a batch queue of submitted jobs
- provides an interface for the users of the system
- the master runs the middleware needed for the execution of programs and management of the cluster

- middleware is formed by the libraries for executing parallel programs.

- many of these libraries effectively provide only advanced message-based communication facilities, but are not capable of handling faulty processes, security, etc.

o compute nodes - often need nothing else but a standard operating system.

An example of a cluster computing system.



▪ Alternative to hierarchical organization - symmetric approach is followed in the MOSIX system

o MOSIX attempts to provide a single-system image of a cluster - to a process, a cluster computer appears to be a single computer.

○ providing such an image under all circumstances is impossible.

○ In MOSIX, the high degree of transparency is provided by allowing processes to dynamically and preemptively migrate between the nodes that make up the cluster.

○ Process migration allows a user to start an application on any node (referred to as the home node), after which it can transparently move to other nodes, for example, to make efficient use of resources.

Grid Computing Systems

▪ Grid computing systems have a high degree of heterogeneity: no assumptions are made concerning hardware, operating systems, networks, administrative domains, security policies, etc.

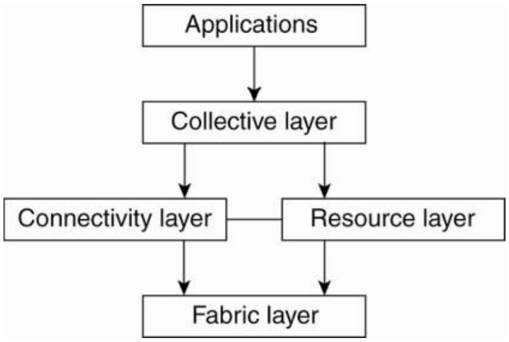
▪ Key issue in a grid computing system - resources from different organizations are brought together to allow the collaboration of a group of people or institutions - forms a virtual organization.

○ Members of the same virtual organization have access rights to the resources that are provided to that organization.

○ Resources consist of compute servers (including supercomputers, possibly implemented as cluster computers), storage facilities, and databases. In addition, special networked devices such as telescopes, sensors, etc., can be provided as well.

▪ Software required for grid computing evolves around providing access to resources from different administrative domains, and to only those users and applications that belong to a specific virtual organization.

A layered architecture for grid computing systems.



Four layer architecture:

1. fabric layer - provides interfaces to local resources at a specific site.

○ nterfaces are tailored to allow sharing of resources within a virtual organization.

○ provide functions for querying the state and capabilities of a resource, along with functions for actual resource management (e.g., locking resources).

2a. connectivity layer - consists of communication protocols for supporting grid transactions that span the usage of multiple resources.

○ contains security protocols to authenticate users and resources.

○ in many cases human users are not authenticated - programs acting on behalf of the users are authenticated.

2b. resource layer - responsible for managing a single resource.

○ uses functions provided by the connectivity layer and calls directly the interfaces made available by the fabric layer.

○ responsible for access control, and hence will rely on the authentication performed as part of the connectivity layer.

3. collective layer - handles access to multiple resources

○ consists of services for resource discovery, allocation and scheduling of tasks onto multiple resources, data replication, etc.

○ may consist of many different protocols for many different purposes, reflecting the broad spectrum of services it may offer to a virtual organization.

4. application layer - consists of the applications that operate within a virtual organization and which make use of the grid computing environment.

Grid Middleware layer - collective, connectivity, and resource layers.

○ provide access to and management of resources that are potentially dispersed across multiple sites.

○ shift toward a service-oriented architecture in which sites offer access to the various layers through a collection of Web services

○ led to the definition of an alternative architecture known as the Open Grid Services Architecture (OGSA).

○ consists of various layers and many components, making it rather complex.

Distributed Information Systems

○ Evolved in organizations that were confronted with a wealth of networked applications, but for which interoperability turned out to be problematic.

○ Many of the existing middleware solutions are the result of working with an infrastructure in which it was easier to integrate applications into an enterprise-wide information system.

Several levels at which integration took place:

1. a networked application simply consisted of a server running that application (often including a database) and making it available to remote programs, called clients.

○ such clients could send a request to the server for executing a specific operation, after which a response would be sent back. Integration at the lowest level would allow clients to wrap a number of requests, possibly for different servers, into a single larger request and have it executed as a distributed transaction.

○ The key idea was that all, or none of the requests would be executed.

2. As applications became more sophisticated and were gradually separated into independent components (notably distinguishing database components from processing components), it became clear that integration should also take place by letting applications communicate directly with each other.

Results: two forms of distributed systems

○ transaction processing systems

○ enterprise application integration (EAI)

Transaction Processing Systems

- Focus on database applications - operations on a database are usually carried out in the form of transactions.
- Programming using transactions requires special primitives that must either be supplied by the underlying distributed system or by the language runtime system.
- Example primitives for transactions

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

- Exact list of primitives depends on what kinds of objects are being used in the transaction.

e.g. mail system - may be primitives to send, receive, and forward mail.

e.g. accounting system - may be different.

- Ordinary statements, procedure calls, etc, are allowed inside a transaction.
- Remote procedure calls (RPCs) procedure calls to remote servers, are often also encapsulated in a transaction, leading to what is known as a transactional RPC.

Properties of transactions (ACID):

1. **A**tomic: To the outside world, the transaction happens indivisibly.

- ensures that each transaction either happens completely, or not at all
- if it happens, it happens in a single indivisible, instantaneous action.
- while a transaction is in progress, other processes (whether or not they are themselves involved in transactions) cannot see any of the intermediate states.

2. **C**onsistent: The transaction does not violate system invariants.

- if the system has certain invariants that must always hold, if they held before the transaction, they will hold afterward too

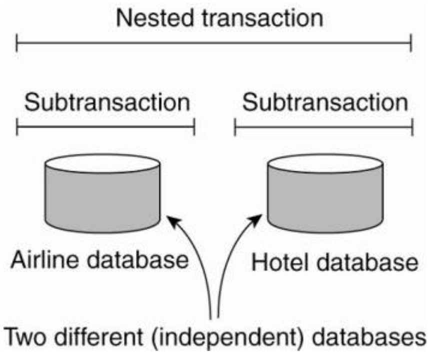
e.g. a banking system - a key invariant is the law of conservation of money. After every internal transfer, the amount of money in the bank must be the same as it was before the transfer

3. **I**solated: Concurrent transactions do not interfere with each other.

- transactions are isolated or serializable
- if two or more transactions are running at the same time, to each of them and to other processes, the final result looks as though all transactions ran sequentially in some (system dependent) order.

4. **D**urable: Once a transaction commits, the changes are permanent.

- once a transaction commits, no matter what happens, the transaction goes forward and the results become permanent.
- no failure after the commit can undo the results or cause them to be lost
- A nested transaction is constructed from a number of subtransactions,



- The top-level transaction may fork off children that run in parallel with one another, on different machines, to gain performance or simplify programming.
- Each of these children may also execute one or more subtransactions, or fork off its own children.

Problems of Subtransactions: permanence applies only to top-level transactions.

e.g.

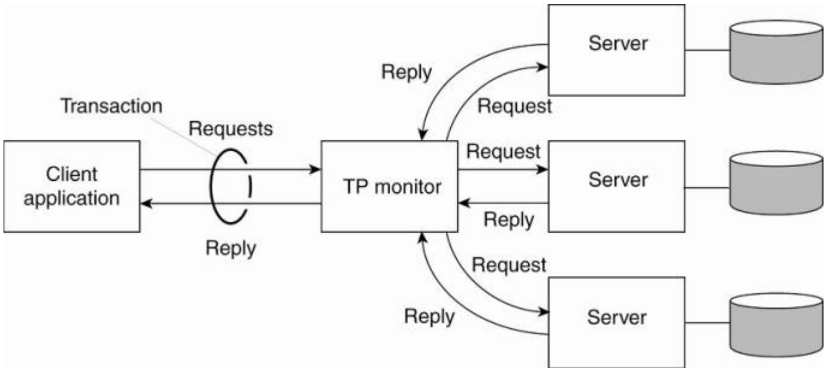
- a transaction starts several subtransactions in parallel, and one of these commits, making its results visible to the parent transaction.
- after further computation, the parent aborts, restoring the entire system to the state it had before the top-level transaction started
- the results of the subtransaction that committed must nevertheless be undone.

Solution:

When any transaction or subtransaction starts, it is conceptually given a private copy of all data in the entire system for it to manipulate as it wishes.

- If it aborts, its private universe just vanishes, as if it had never existed.
- If it commits, its private universe replaces the parent's universe.
- Thus if a subtransaction commits and then later a new subtransaction is started, the second one sees the results produced by the first one. Likewise, if an enclosing (higher-level) transaction aborts, all its underlying subtransactions have to be aborted as well.

- Nested transactions provide a natural way of distributing a transaction across multiple machines.
- They follow a logical division of the work of the original transaction.
- Each subtransaction can be managed separately and independenty.
- Early enterprise middleware systems handled distributed (or nested) transactions using a transaction processing monitor or TP monitor for integrating applications at the server or database level.
- Its main task was to allow an application to access multiple server/databases by offering it a transactional programming model
- The role of a TP monitor in distributed systems.



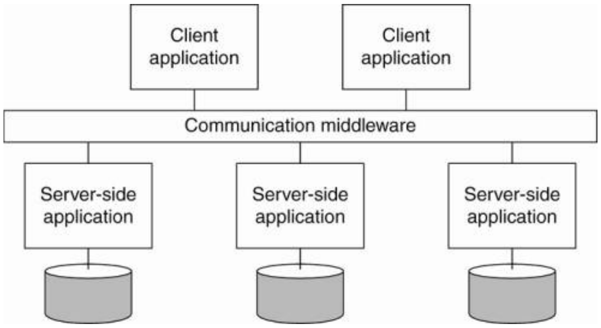
Enterprise Application Integration

- The more applications became decoupled from the databases they were built upon, the more evident it became that facilities were needed to integrate applications independent from their databases.

- Application components should be able to communicate directly with each other and not merely by means of the request/reply behavior that was supported by transaction processing systems.

- Result:

Middleware as a communication facilitator in enterprise application integration.



Several types of communication middleware:

1. Remote procedure calls (RPC) -

- an application component can send a request to another application component by doing a local procedure call, which results in the request being packaged as a message and sent to the callee.

- the result will be sent back and returned to the application as the result of the procedure call.

2. Remote method invocations (RMI) –

- An RMI is the same as an RPC, except that it operates on objects instead of applications.

Problems with RPC and RMI:

- The caller and callee both need to be up and running at the time of communication.

- They need to know exactly how to refer to each other.

Results:

- Message-oriented middleware (MOM) - applications send messages to logical contact points, often described by means of a subject.

- Publish/subscribe systems - applications can indicate their interest for a specific type of message, after which the communication middleware will take care that those messages are delivered to those applications.

Distributed Pervasive Systems

Above distributed systems characterized by their stability:

- nodes are fixed and have a more or less permanent and high-quality connection to a network.

Mobile and embedded computing devices:

- instability is the default behavior

Distributed pervasive system - is part of the surroundings -> inherently distributed.

Features:

- general lack of human administrative control

- devices can be configured by their owners

- they need to automatically discover their environment and fit in as best as possible. This nestling in has been made more precise by Grimm et al. (2004) by formulating the following

Three requirements for pervasive applications (Grimm et al. 2004) :

❖ Embrace contextual changes.

- a device must be continuously be aware of the fact that its environment may change all the time

❖ Encourage ad hoc composition.

- many devices in pervasive systems will be used in very different ways by different users – make it easy to configure the suite of applications running on a device

❖ Recognize sharing as the default.

- devices generally join the system in order to access (and possibly provide) information

Examples:

Home Systems

- built around home networks
- consist of one or more personal computers
- integrate consumer electronics such as TVs, audio and video equipment, gaming devices, (smart) phones, PDAs, and other personal wearables into a single system.
- Now/Soon: all kinds of devices such as kitchen appliances, surveillance cameras, clocks, controllers for lighting, and so on, will all be hooked up into a single distributed system.

Several challenges:

- System should be completely self-configuring and self-managing

e.g. Universal Plug and Play (UPnP) standards by which devices automatically obtain IP addresses, can discover each other, etc. (UPnP Forum).

- Unclear how software and firmware in devices can be easily updated without manual intervention, or when updates do take place, that compatibility with other devices is not violated.

Managing "personal space."

- A home system consists of many shared as well as personal devices
- Data in a home system is also subject to sharing restrictions.

Electronic Health Care Systems

- New devices are being developed to monitor the well-being of individuals and to automatically contact physicians when needed.
- Major goal is to prevent people from being hospitalized.
- Personal health care systems rea equipped with various sensors organized in a (preferably wireless) body-area network (BAN).
- Such a network should at worst only minimally hinder a person.

Organization:

1. A central hub is part of the BAN and collects data as needed. Data is then offloaded to a larger storage device.

- Advantage: the hub can also manage the BAN.

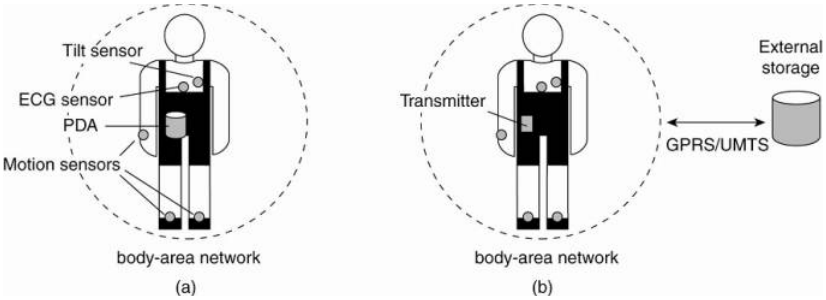
2. The BAN is continuously hooked up to an external network through a wireless connection, to which it sends monitored data.

- Separate techniques will need to be deployed for managing the BAN.
- Further connections to a physician or other people may exist as well.

Monitoring a person in a pervasive electronic health care system, using

(a) a local hub o

(b) a continuous wireless connection.



Questions:

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

For reasons of efficiency:

- devices and body-area networks will be required to support in-network data processing
- monitoring data will have to be aggregated before permanently storing it or sending it to a physician.

Unlike distributed information systems, there is yet no clear answer to these questions.

Sensor Networks

- Part of the enabling technology for pervasiveness
- In virtually all cases they are used for processing information.

➔They do more than just provide communication services as with traditional computer networks. (Akyildiz et al. 2002)

- A sensor network consists of tens to hundreds or thousands of relatively small nodes, each equipped with a sensing device.
- Most sensor networks use wireless communication, and the nodes are often battery powered.
- Their limited resources, restricted communication capabilities, and constrained power consumption demand that efficiency be high on the list of design criteria.

- The relation with distributed systems can be made clear by considering sensor networks as distributed databases.
- To organize a sensor network as a distributed database, there are essentially two extremes:

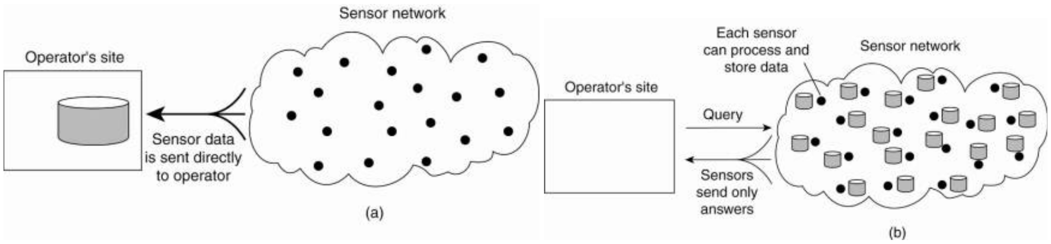
1. Sensors do not cooperate but simply send their data to a centralized database located at the operator's site.

2. Forward queries to relevant sensors and to let each compute an answer, requiring the operator to sensibly aggregate the returned answers.

Organizing a sensor network database, while storing and processing data:

(a) only at the operator's site

(b) only at the sensors.


Sensor network
Operator's site
Sensor data is sent directly to operator
(a)
Each sensor can process and store data
Sensor network
Operator's site
Query
Sensors send only answers
(b)

Neither solution is attractive:

- The 1st requires that sensors send all their measured data through the network, which may waste network resources and energy.

- The 2nd may also be wasteful as it discards the aggregation capabilities of sensors which would allow much less data to be returned to the operator. What is needed are facilities for in-network data processing, as we also encountered in pervasive health care systems.

In-network processing can be done by:

- Forwarding a query to all sensor nodes along a tree encompassing all nodes and to subsequently aggregate the results as they are propagated back to the root, where the initiator is located.

- Aggregation will take place where two or more branches of the tree come to together.

- Solution introduces questions:

1. How do we (dynamically) set up an efficient tree in a sensor network?

2. How does aggregation of results take place? Can it be controlled?

3. What happens when network links fail?

Resolution:

TinyDB - implements a declarative (database) interface to wireless sensor networks.

TinyDB can use any tree-based routing algorithm.

- An intermediate node will collect and aggregate the results from its children, along with its own findings, and send that toward the root.

§ To increase efficiency, queries span a period of time allowing for careful scheduling of operations so that network resources and energy are optimally consumed.

§ When queries can be initiated from different points in the network, using single-rooted trees such as in TinyDB may not be efficient enough.

Alternative - sensor networks may be equipped with special nodes where results are forwarded to, as well as the queries related to those results.

- e.g, queries and results related temperature readings are collected at a different location than those related to humidity measurements. This approach corresponds directly to the notion of publish/subscribe systems