Types of Distributed Systems

Types of Distributed Systems

1. Cluster computing

- In 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.
- In addition, each node runs the same operating system.
- A characteristic feature of cluster computing is its homogeneity.
- In most cases, the computers in a cluster are largely the same, they all have the same operating system, and are all connected through the same network.

Cluster Computing Systems

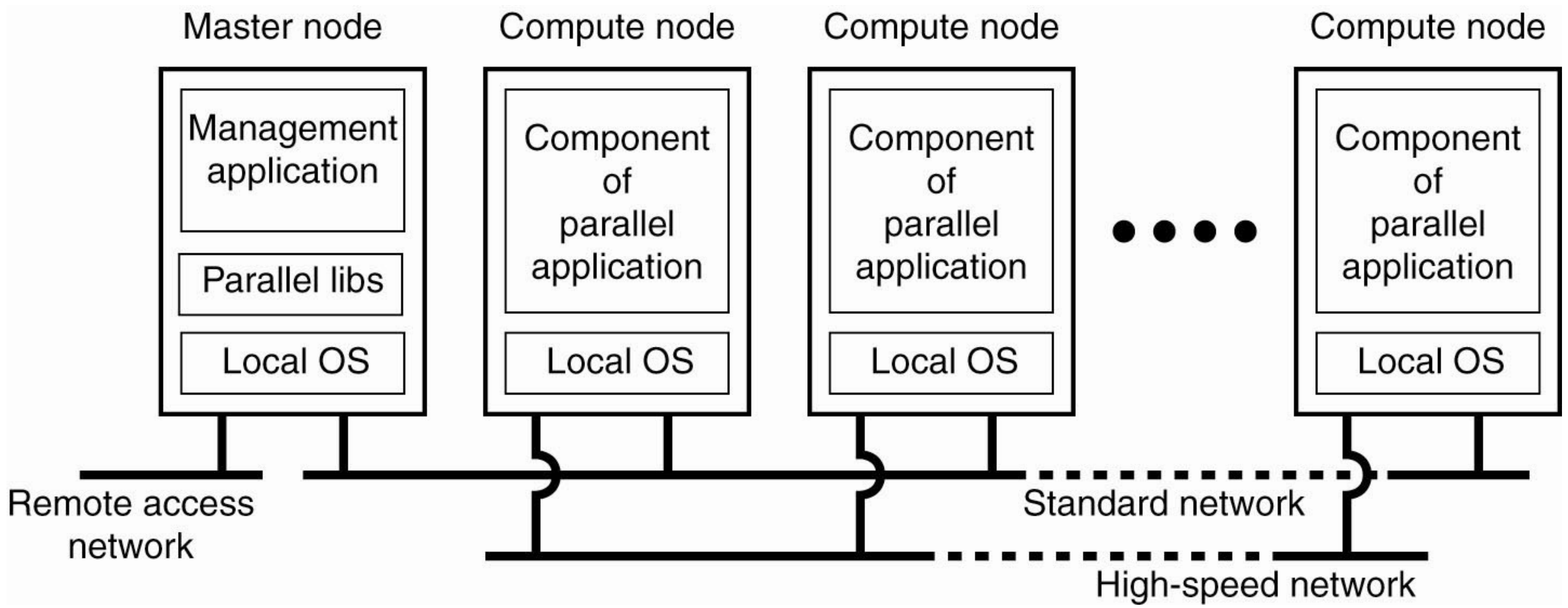
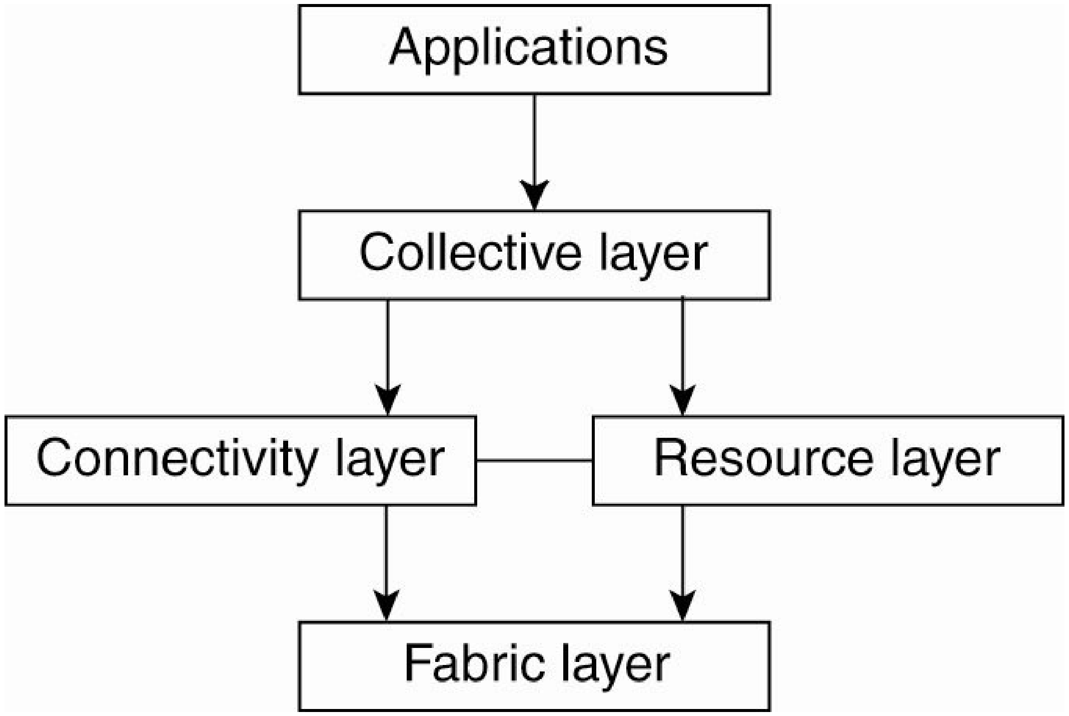


Figure 1-6. An example of a cluster computing system.

2. Grid Computing Systems



A subgroup consists of distributed systems that are often 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***.

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

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

2. Grid Computing Systems

➔The lowest ***fabric layer*** provides interfaces to local resources at a specific site. Typically, they will provide functions for querying the state and capabilities of a resource, along with functions for actual resource management (e.g., locking resources).

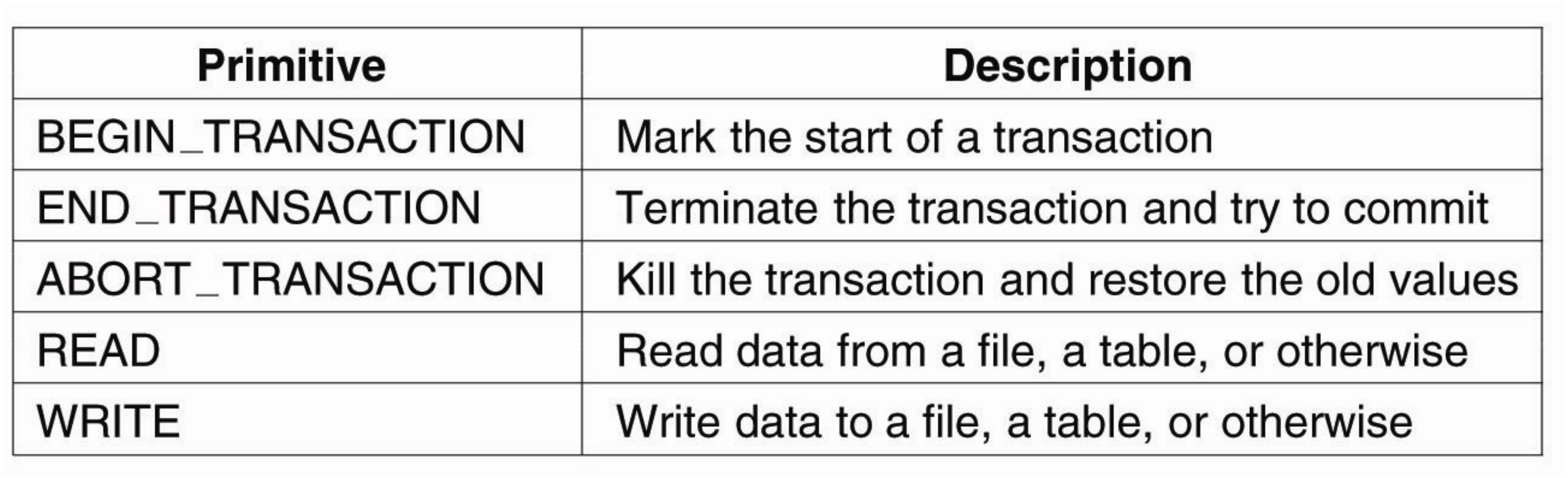
➔The ***connectivity layer*** consists of communication protocols for supporting grid transactions that span the usage of multiple resources. For example, protocols are needed to transfer data between resources, or to simply access a resource from a remote location.

➔The ***resource layer*** is responsible for managing a single resource. It uses the functions provided by the connectivity layer and calls directly the interfaces made available by the fabric layer.

➔ The ***collective layer*** deals with handling access to multiple resources and typically consists of services for resource discovery, allocation and scheduling of tasks onto multiple resources, data replication, and so on.

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

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)

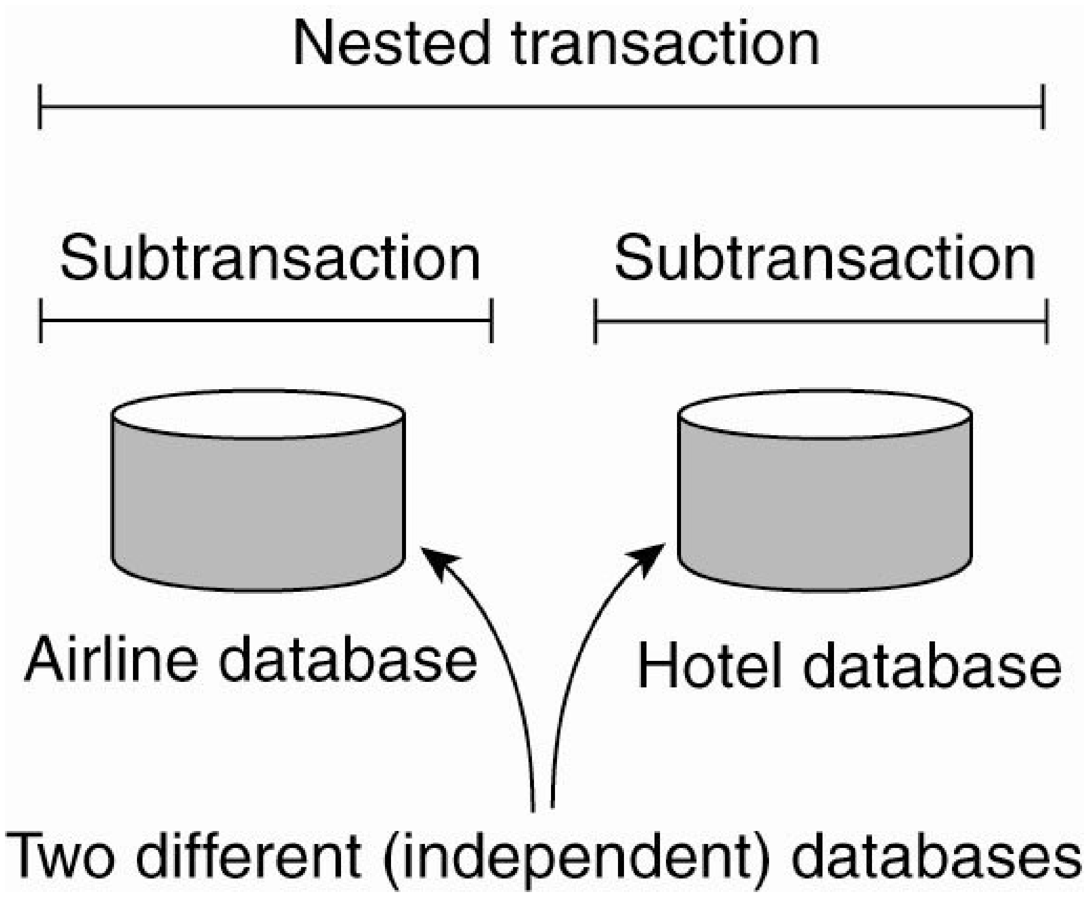


Figure 1-9. A nested transaction.

Transaction Processing Systems (3)

➔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.

➔Subtransactions give rise to a subtle, but important, problem. Imagine that 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.

➔Consequently, the results of the subtransaction that committed must nevertheless be undone. Thus the permanence referred to above applies only to top-level transactions.

➔Since transactions can be nested arbitrarily deeply, considerable administration is needed to get everything right. The semantics are clear, however.

➔Nested transactions are important in distributed systems, for they provide a natural way of distributing a transaction across multiple machines.

➔They follow a logical division of the work of the original transaction.

Transaction Processing Systems (4)

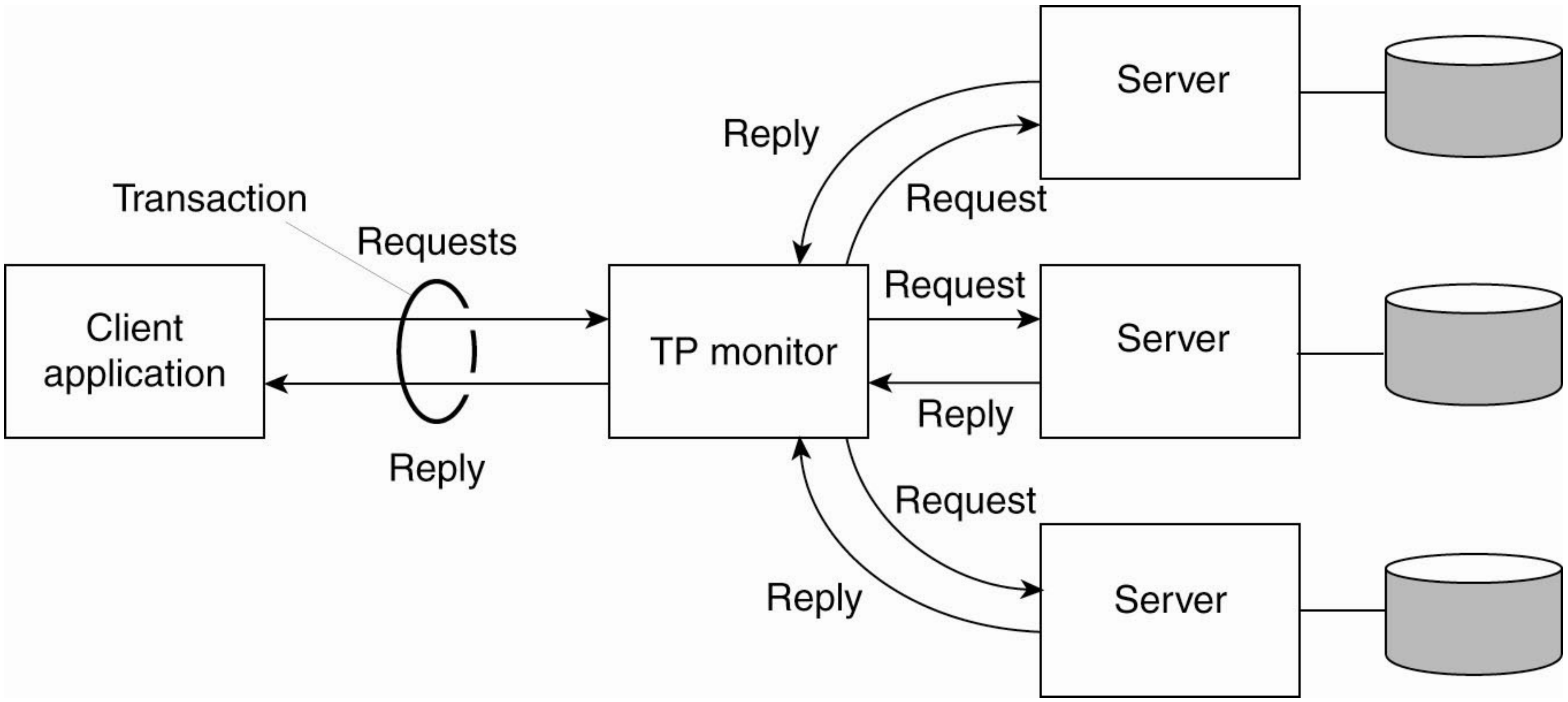


Figure 1-10. The role of a TP monitor in distributed systems.

Transaction Processing Systems (4)

➔In the early days of enterprise middleware systems, the component that handlled distributed (or nested) transactions formed the core for integrating applications at the server or database level.

➔This component was called a transaction processing monitor or TP monitor for short. Its main task was to allow an application to access multiple server/databases by offering it a transactional programming model.

Enterprise Application Integration

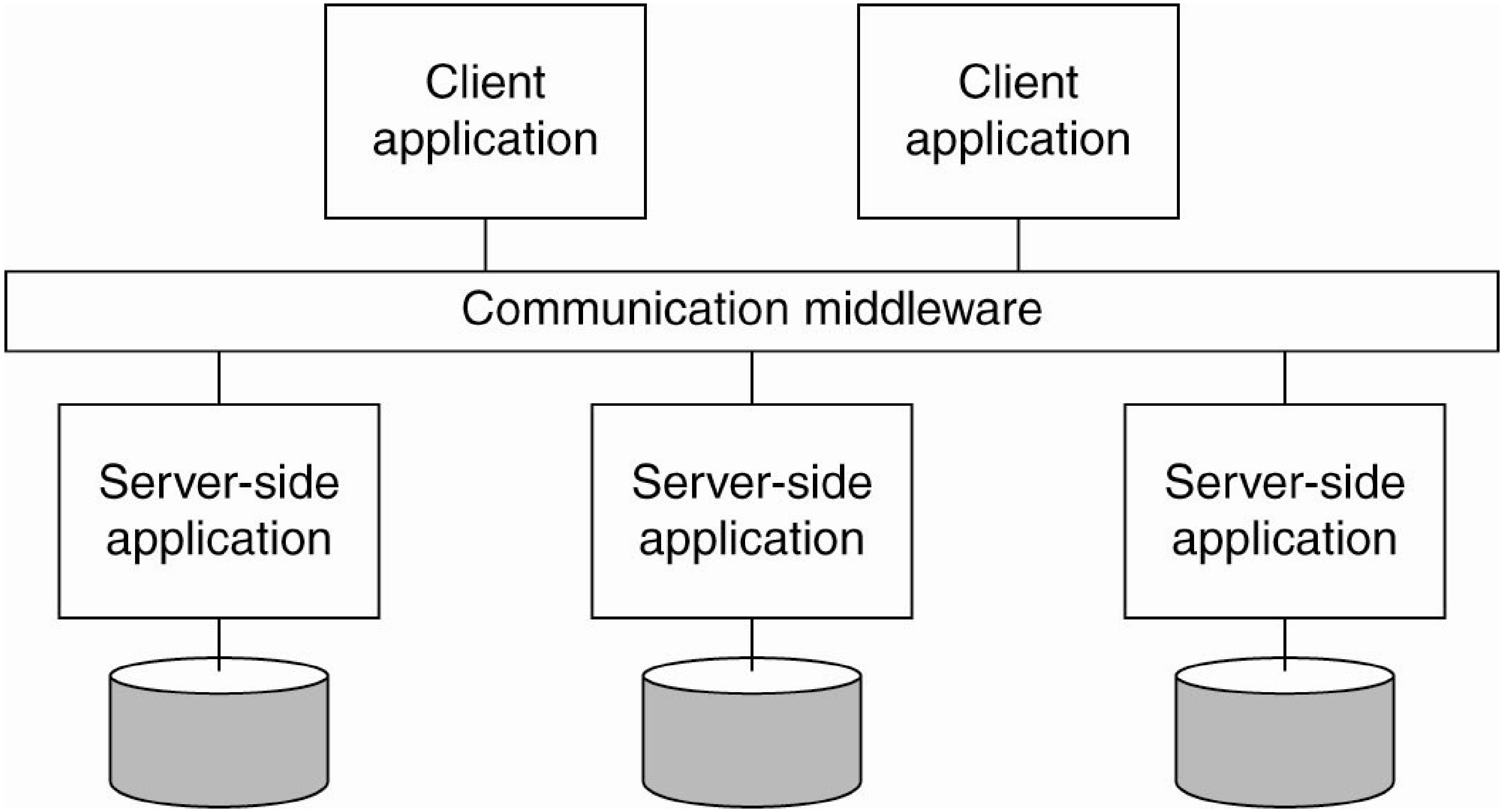


Figure 1-11. Middleware as a communication facilitator in enterprise application integration.

Enterprise Application Integration

- Several types of communication middleware exist. With remote procedure calls (RPC), an application component can effectively 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.
- Likewise, the result will be sent back and returned to the application as the result of the procedure call.
- As the popularity of object technology increased, techniques were developed to allow calls to remote objects, leading to what is known as remote method invocations (RMI).
- An RMI is essentially the same as an RPC, except that it operates on objects instead of applications.

Distributed Pervasive Systems

Requirements for pervasive systems

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

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)

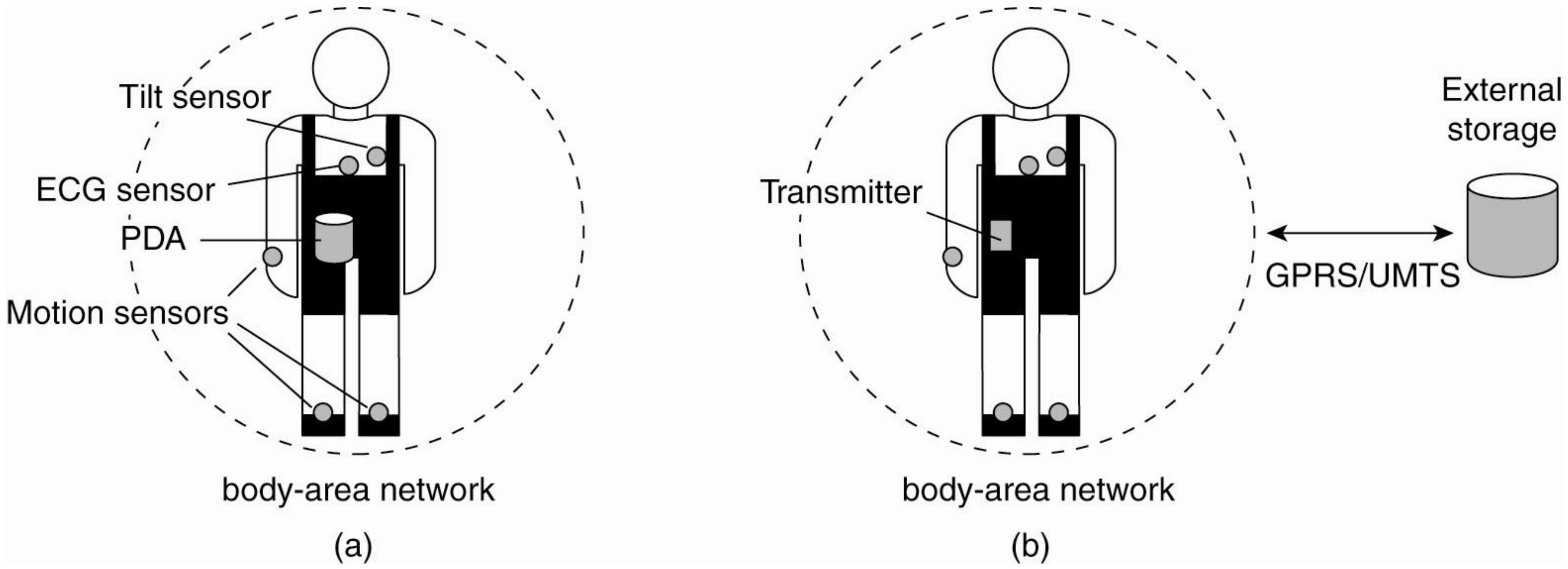


Figure 1-12. Monitoring a person in a pervasive electronic health care system, using (a) a local hub or (b) a continuous wireless connection.

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)

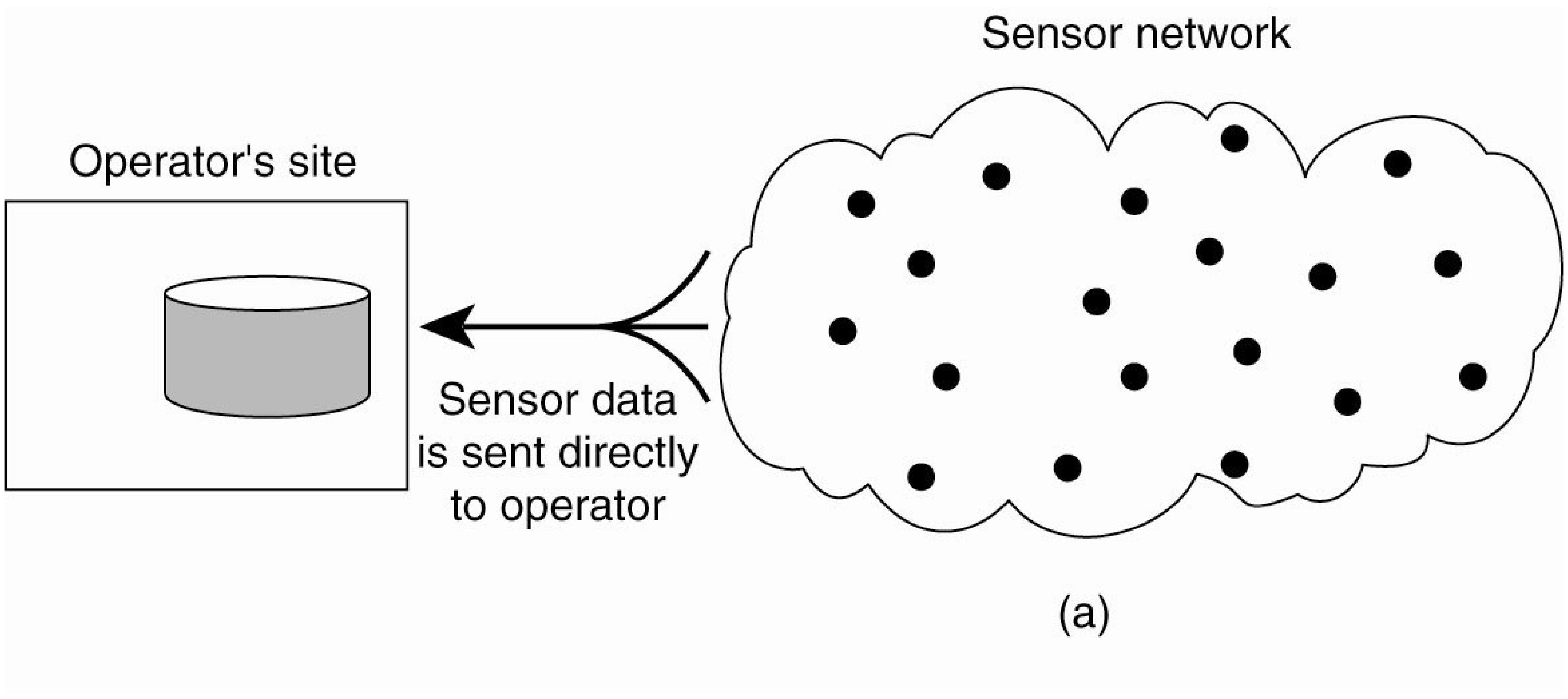


Figure 1-13. Organizing a sensor network database, while storing and processing data (a) only at the operator’s site or …

Sensor Networks (3)

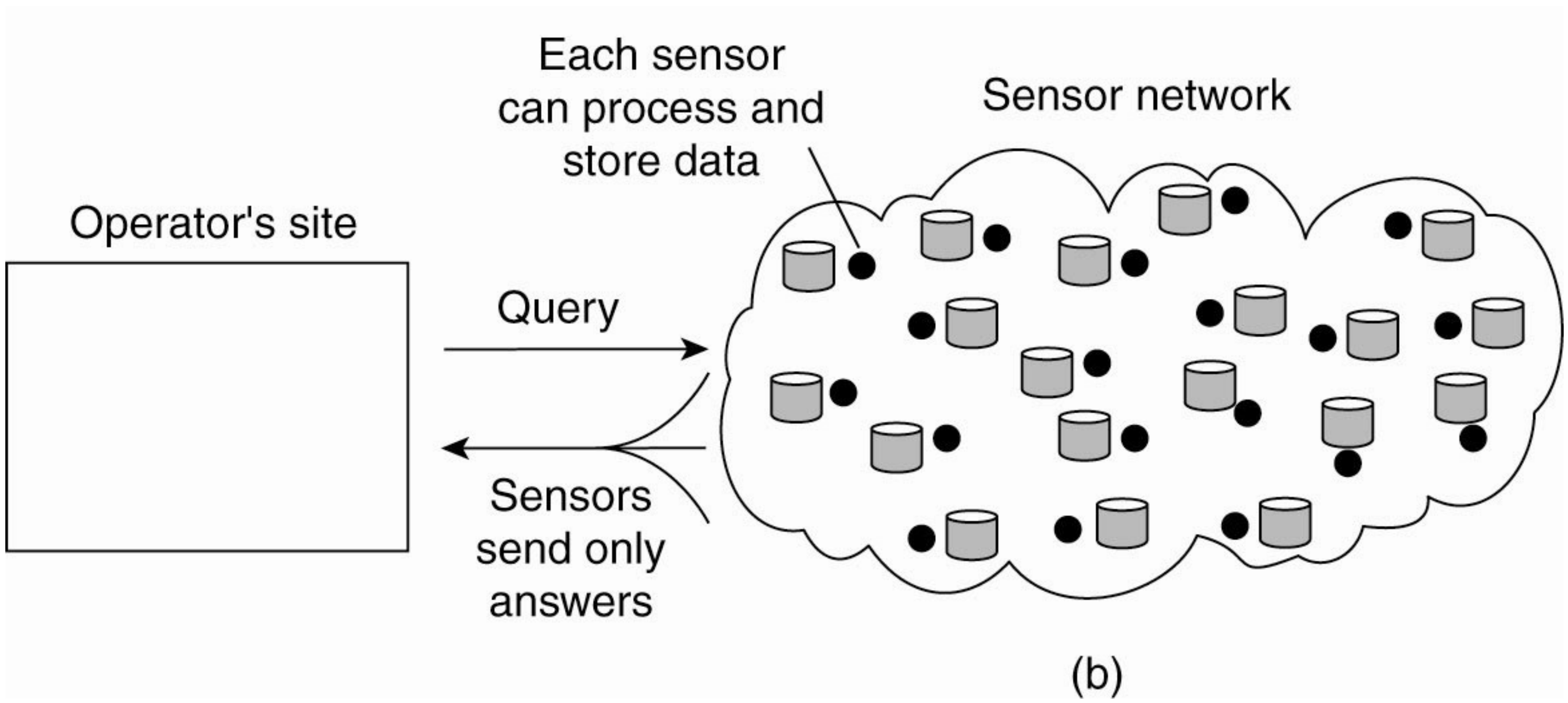

Each sensor can process and store data
Sensor network
Operator's site
Query
Sensors send only answers
(b)

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