Distributed Systems

Mahmoud Abou El-Magd Soliman
Department of Computer Science
Faculty of Computers and Artificial Intelligence
Sohag University

Chapter one

Introduction to Distributed Systems

Introduction

Computer systems are undergoing a revolution. From 1945, when the modem computer era began, until about 1985, computers were large and expensive. Even minicomputers cost at least tens of thousands of dollars each. As a result, most organizations had only a handful of computers, and for lack of a way to connect them, these operated independently from one another.

Introduction

Starting around the mid-1980s, however, two advances in technology began to change that situation. The first was the development of powerful microprocessors. Many of these had the computing power of a mainframe (i.e., large) computer, but for a fraction of the price.

The second development was the invention of high-speed computer networks. Local-area networks or LANs allow hundreds of machines within a building to be connected in such a way that small amounts of information can be transferred between machines in a few microseconds or so.

Introduction

The result of these technologies is that it is now not only feasible, but easy, to put together computing systems composed of large numbers of computers connected by a high-speed network. They are usually called computer networks or distributed systems, in contrast to the previous centralized systems (or single processor systems) consisting of a single computer, its peripherals, and perhaps some remote terminals.

Definition of a Distributed System

Various definitions of distributed systems have been given in the literature:

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

Note that no assumptions are made concerning the type of computers. In principle, even within a single system, they could range from high-performance mainframe computers to small nodes in sensor networks.

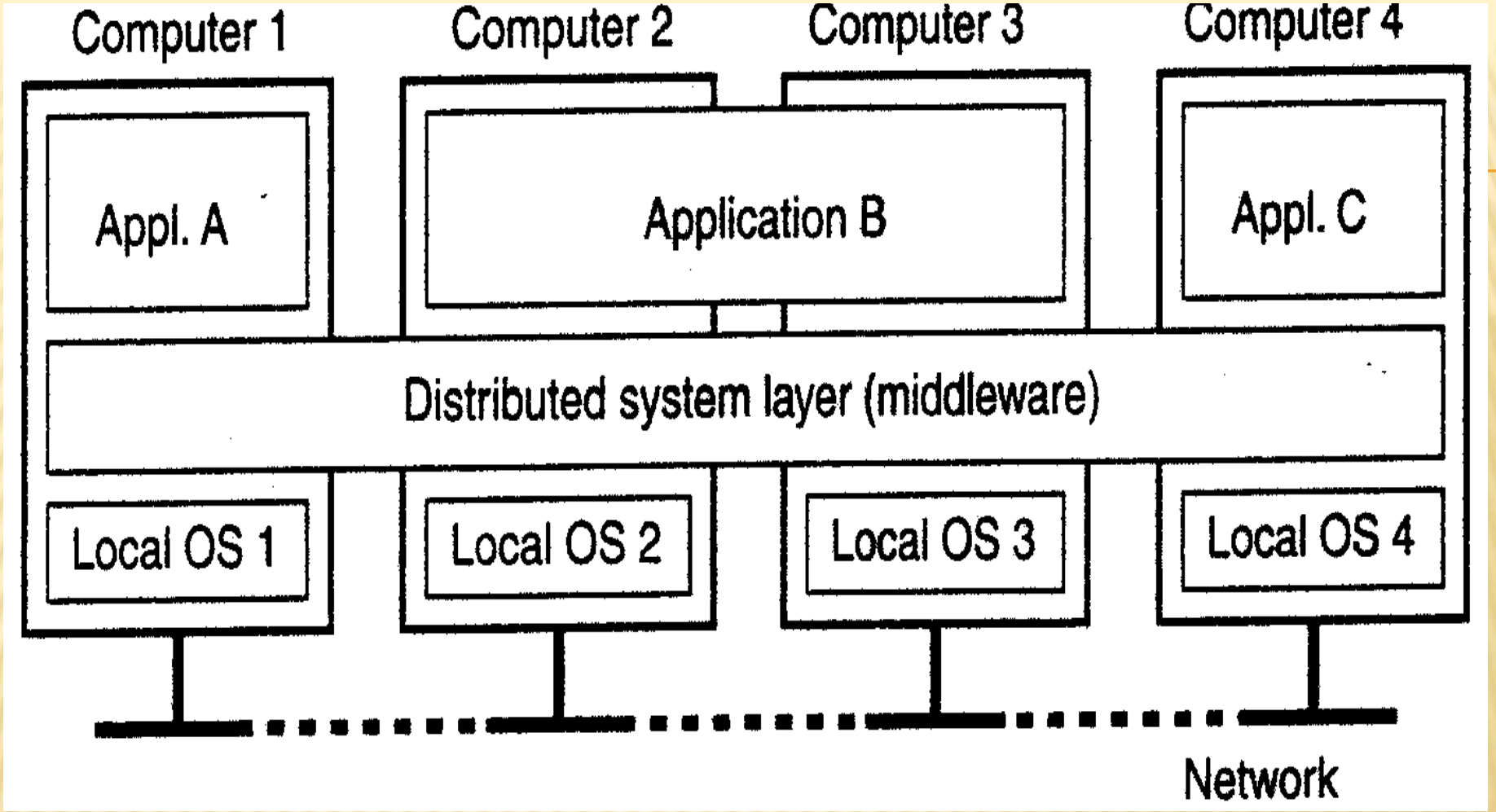
Characteristics of Distributed Systems

- One important characteristic is that differences between the various computers and the ways in which they communicate are mostly hidden from users.
- Another important characteristic is that users and applications can interact with a distributed system in a consistent and uniform way, regardless of where and when interaction takes place.

Characteristics of Distributed Systems

- In principle, distributed systems should also be relatively easy to expand or scale.
- A distributed system will normally be continuously available, although perhaps some parts may be temporarily out of order.
- Users and applications should not notice that parts are being replaced or fixed, or that new parts are added to serve more users or applications.

In order to support heterogeneous computers and networks while offering a single-system view, distributed systems are often organized by means of a layer of software-that is, logically placed between a higher-level layer consisting of users and applications, and a layer underneath consisting of operating systems and basic communication facilities, as shown in Fig. 1-1 Accordingly, such a distributed system is sometimes called middleware.



A distributed system organized as middleware. The middleware layer extends over multiple machines, and offers each application the same interface.

Goals

- A distributed system should make resources easily accessible
- A Distributed system should reasonably hide the fact that resources are distributed across a network
- A Distributed system should be open
- A Distributed system should be scalable

Making Resources Accessible

The main goal of a distributed system is to make it easy for the users (and applications) to access remote resources, and to share them in a controlled and efficient way. There are many reasons for wanting to share resources. One obvious reason is that of economics. For example, it is cheaper to let a printer be shared by several users in a small office than having to buy and maintain a separate printer for each user. Likewise, it makes economic sense to share costly resources such as supercomputers, high-performance storage systems, and other expensive peripherals.

Making Resources Accessible

Connecting users and resources also makes it easier to collaborate and exchange information. The connectivity of the Internet is now leading to numerous virtual organizations in which geographically widely-dispersed groups of people work together by means of groupware, that is, software for collaborative editing, teleconferencing, and so on. Likewise, the Internet connectivity has enabled electronic commerce allowing us to buy and sell all kinds without actually having to go to a store or even leave home.

Making Resources Accessible

However, as connectivity and sharing increase, security is becoming increasingly important.

Another security problem is that of tracking communication to build up a preference profile of a specific user. A related problem is that increased connectivity can also lead to unwanted communication, such as electronic junk mail, often called spam.

Distribution Transparency

An important goal of a distributed system is to hide the fact that its processes and resources are physically distributed across multiple computers. A distributed system that is able to present itself to users and applications as if it were only a single computer system is said to be transparent. Let us first take a look at what kinds of transparency exist in distributed systems.

Types of Transparency

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

Different forms of transparency in a distributed system.

Access transparency

Access transparency deals with hiding differences in data representation and the way that resources can be accessed by users. At a basic level, we wish to hide differences in machine architectures, but more important is that we reach agreement on how data is to be represented by different machines and operating systems. For example, a distributed system may have computer systems that run different operating systems, each having their own file-naming conventions

Location transparency

Location transparency refers to the fact that users cannot tell where a resource is physically located in the system. In particular, location transparency can be achieved by assigning only logical names to resources, that is, names in which the location of a resource is not secretly encoded. An example of a such a name is the URL http://www.prenhall.com/index.html. which gives no clue about the location of Prentice Hall's main Web server.

Transparency

Distributed systems in which resources can be moved without affecting how those resources can be accessed are said to provide migration transparency.

Even stronger is the situation in which resources can be relocated while they are being accessed without the user or application noticing anything. In such cases, the system is said to support relocation transparency.

Transparency

As we shall see, replication plays a very important role in distributed systems. For example, resources may be replicated to increase availability or to improve performance by placing a copy close to the place where it is accessed. Replication transparency deals with hiding the fact that several copies of a resource exist. To hide replication from users, it is necessary that all replicas have the same name.

Concurrency Transparency

In many cases, sharing resources is done in a cooperative way. However. there are also many examples of competitive sharing of resources. For example, two independent users may each have stored their files on the same file server or may be accessing the same tables in a shared database. In such cases, it is important that each user does not notice that the other is making use of the same resource. This phenomenon is called concurrency transparency.

Failure Transparency

Making a distributed system failure transparent means that a user does not notice that a resource (he has possibly never heard of) fails to work properly, and that the system subsequently recovers from that failure. Masking failures is one of the hardest issues in distributed systems and is even impossible when certain apparently realistic assumptions are made.

Openness

Another important goal of distributed systems is openness. An open distributed system is a system that offers services according to standard rules. For example, in computer networks, standard rules govern the format, contents, and meaning of messages sent and received. Such rules are formalized in protocols. In distributed systems, services are generally specified through interfaces. In other words, they specify the names of the functions that are available together with types of the parameters, return values, possible exceptions that can be raised, and so on.

Scalability

Scalability of a system can be measured along at least three different dimensions. First, a system can be scalable with respect to its size, meaning that we can easily add more users and resources to the system. Second, a geographically scalable system is one in which the users and resources may lie far apart. Third, a system can be administratively scalable, that it can still be easy to manage even if it spans many independent administrative organizations.

Scalability Problems

Let us first consider scaling with respect to size. If more users or resources need to be supported, we are often confronted with the limitations of centralized services, data, and algorithms. For example, many services are centralized in the sense that they are implemented by means of only a single server running on a specific machine in the distributed system. The problem with this scheme is obvious: the server can become a bottleneck as the number of users and applications grows. Even if we have virtually unlimited processing and storage capacity, communication with that server will eventually prohibit further growth.

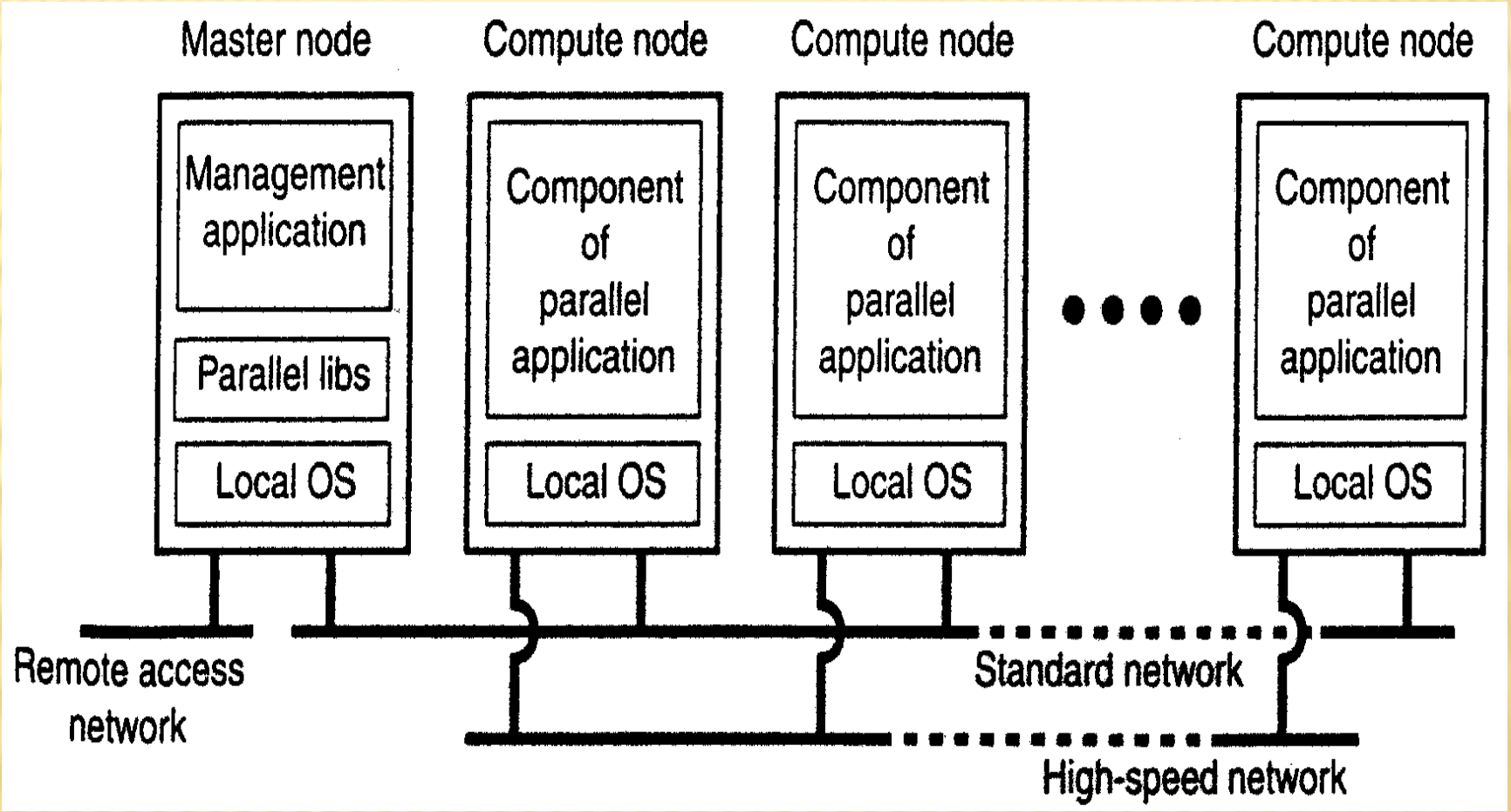
Scalability Problems

Unfortunately. using only a single server is sometimes unavoidable. Imagine that we have a service for managing highly confidential information such as medical records, bank accounts. and so on. In such cases, it may be best to implement that service by means of a single server in a highly secured separate room, and protected from other parts of the distributed system through special network components. Copying the server to several locations to enhance performance maybe out of the question as it would make the service less secure.

Types of Distributed Systems

An important class of distributed systems is the one used for high-performance computing tasks. Roughly speaking, one can make a distinction between two subgroups. In cluster computing the underlying hardware consists of a collection of similar workstations or PCs, closely connected by means of a highspeed local-area network. In addition, each node runs the same operating system.

Cluster Computing



An example of a cluster computing system.

Types of Distributed Systems

The situation becomes quite different in the case of grid computing (Cloud Computing). This 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.

Distributed Information Systems

In many cases, 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.

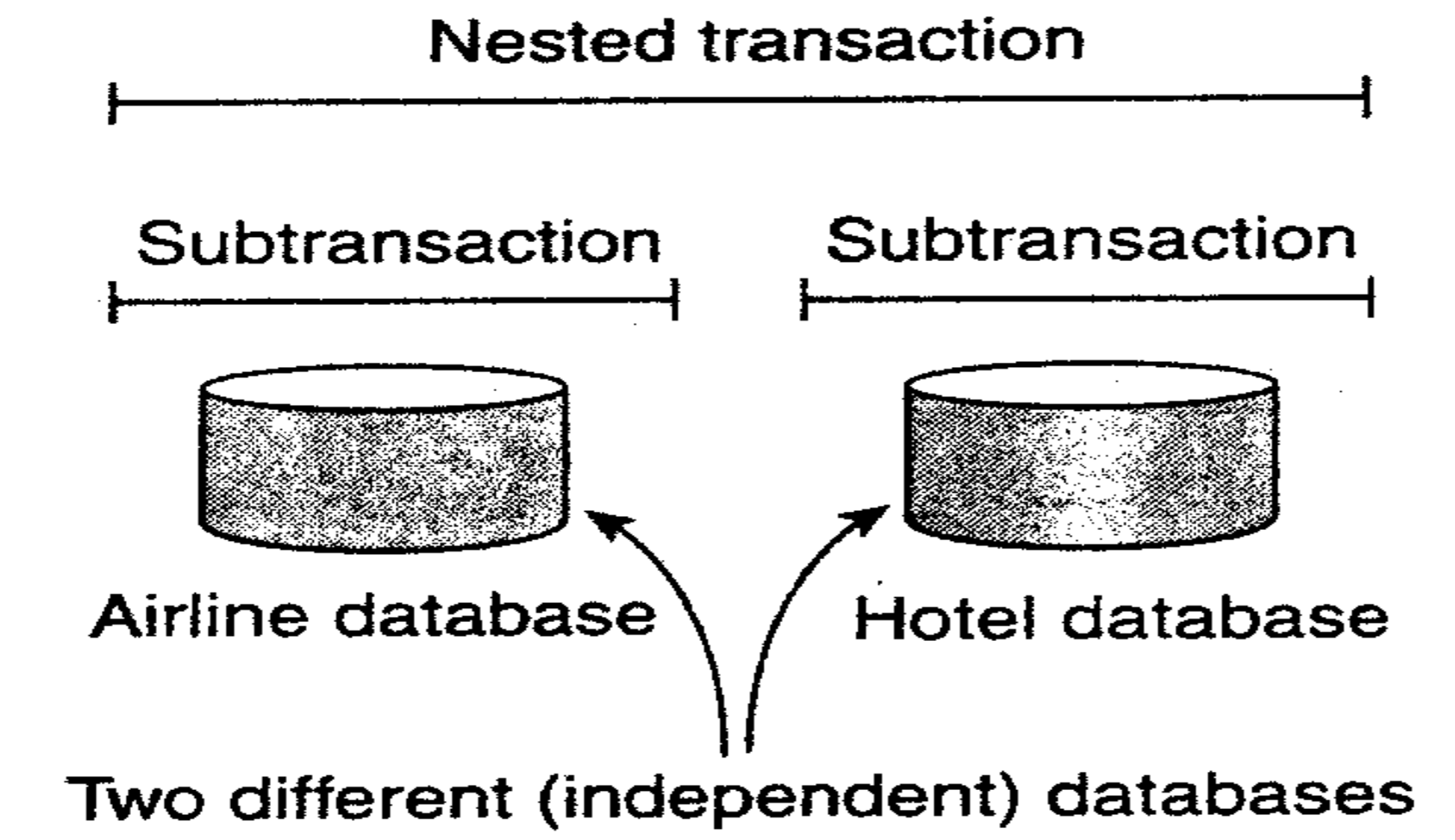
Transaction Processing Systems

To clarify our discussion, let us concentrate on database applications. In practice, operations on a database are usually carried out in the form of transactions. In a mail system, there might be primitives to send, receive, and forward mail. In an accounting system, they might be quite different. READ and WRITE are typical examples, however.

Transaction Processing Systems

So far, transactions have been defined on a single database. A nested transaction is constructed from a number of sub-transactions, as shown in Figure. 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 sub-transactions.

Transaction Processing Systems



A nested transaction.