UNIT – I

LESSON 1: DISTRIBUTED SYSTEMS

CONTENTS

1.0 Aim and Objectives
1.1. Introduction
1.2. Organization
1.3. Goals and Advantages
1.4. Disadvantages
1.5. Architecture
1.6. Concurrency
1.7. Languages
1.8. Let us Sum UP
1.9. Lesson-End Activities
1.10. Points for Discussion
1.11. References

1.0. AIM AND OBJECTIVES

At the end of this Lesson you will be able to

- understand the concept of Distributed Computing,
- organization of Distributed Computing,
- advantages and limitations of Distributed Computing

1.1. INTRODUCTION

Distributed computing is a method of computer processing in which different parts of a program are run simultaneously on two or more computers that are communicating with each other over a network. Distributed computing is a type of **segmented** or parallel computing, but the latter term is most commonly used to refer to processing in which different parts of a program run simultaneously on two or more processors that are part of the same computer. While both types of processing require that a program be segmented—divided into sections that can run simultaneously, distributed computing also requires that the division of the program take into account the different environments on which the different sections of the program will be running. For example, two computers are likely to have different file systems and different hardware components.

An example of distributed computing is BOINC, a framework in which large problems can be divided into many small problems which are distributed to many computers. Later, the small results are reassembled into a larger solution.

Distributed computing is a natural result of using networks to enable computers to communicate efficiently. But distributed computing is distinct from computer networking or **fragmented** computing. The latter refers to two or more computers interacting with each other, but not, typically, sharing the processing of a single program. The World Wide Web is an example of a network, but not an example of distributed computing.

There are numerous technologies and standards used to construct distributed computations, including some which are specially designed and optimized for that purpose, such as Remote Procedure Calls (RPC) or Remote Method Invocation (RMI) or .NET Remoting.

1.2. ORGANIZATION

Organizing the interaction between each computer is of prime importance. In order to be able to use the widest possible range and types of computers, the protocol or communication channel should not contain or use any information that may not be understood by certain machines. Special care must also be taken that messages are indeed delivered correctly and that invalid messages are rejected which would otherwise bring down the system and perhaps the rest of the network.

Another important factor is the ability to send software to another computer in a portable way so that it may execute and interact with the existing network. This may not always be possible or practical when using differing hardware and resources, in which case other methods must be used such as cross-compiling or manually porting this software.

1.3. GOALS AND ADVANTAGES

There are many different types of distributed computing systems and many challenges to overcome in successfully designing one. The main goal of a distributed computing system is to connect users and resources in a transparent, open, and scalable way. Ideally this arrangement is drastically more fault tolerant and more powerful than many combinations of stand-alone computer systems.

Openness

Openness is the property of distributed systems such that each subsystem is continually open to interaction with other systems (see references). Web Services protocols are standards which enable distributed systems to be extended and scaled. In general, an open system that scales has an advantage over a perfectly closed and self-contained system.

Consequently, open distributed systems are required to meet the following challenges:

Monotonicity

Once something is published in an open system, it cannot be taken back.

Pluralism

Different subsystems of an open distributed system include heterogeneous, overlapping and possibly conflicting information. There is no central arbiter of truth in open distributed systems.

Unbounded nondeterminism

Asynchronously, different subsystems can come up and go down and communication links can come in and go out between subsystems of an open distributed system. Therefore the time that it will take to complete an operation cannot be bounded in advance.

1.4. DISADVANTAGES

Technical issues

If not planned properly, a distributed system can decrease the overall reliability of computations if the unavailability of a node can cause disruption of the other nodes. Leslie Lamport famously quipped that: "A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."

Troubleshooting and diagnosing problems in a distributed system can also become more difficult, because the analysis may require connecting to remote nodes or inspecting communication between nodes.

Many types of computation are not well suited for distributed environments, typically owing to the amount of network communication or synchronization that would be required between nodes. If bandwidth, latency, or communication requirements are too significant, then the benefits of distributed computing may be negated and the performance may be worse than a non-distributed environment.

Project-related problems

Distributed computing projects may generate data that is proprietary to private industry, even though the process of generating that data involves the resources of volunteers. This may result in controversy as private industry profits from the data which is generated with the aid of volunteers. In addition, some distributed computing projects, such as biology projects that aim to develop thousands or millions of "candidate molecules" for solving various medical problems, may create vast amounts of raw data. This raw data may be useless by itself without refinement of the raw data or testing of candidate results in real-world experiments. Such refinement and experimentation may be so expensive and time-consuming that it may literally take decades to sift through the data. Until the data is refined, no benefits can be acquired from the computing work.

Other projects suffer from lack of planning on behalf of their well-meaning originators. These poorly planned projects may not generate results that are palpable, or may not generate data that ultimately result in finished, innovative scientific papers. Sensing that a project may not be generating useful data, the project managers may

decide to abruptly terminate the project without definitive results, resulting in wastage of the electricity and computing resources used in the project. Volunteers may feel disappointed and abused by such outcomes. There is an obvious opportunity cost of devoting time and energy to a project that ultimately is useless, when that computing power could have been devoted to a better planned distributed computing project generating useful, concrete results.

Another problem with distributed computing projects is that they may devote resources to problems that may not ultimately be soluble, or to problems that are best pursued later in the future, when desktop computing power becomes fast enough to make pursuit of such solutions practical. Some distributed computing projects may also attempt to use computers to find solutions by number-crunching mathematical or physical models. With such projects there is the risk that the model may not be designed well enough to efficiently generate concrete solutions. The effectiveness of a distributed computing project is therefore determined largely by the sophistication of the project creators.-

1.5. ARCHITECTURE

Various hardware and software architectures are used for distributed computing. At a lower level, it is necessary to interconnect multiple CPUs with some sort of network, regardless of whether that network is printed onto a circuit board or made up of loosely-coupled devices and cables. At a higher level, it is necessary to interconnect processes running on those CPUs with some sort of communication system.

Distributed programming typically falls into one of several basic architectures or categories: Client-server, 3-tier architecture, N-tier architecture, Distributed objects, loose coupling, or tight coupling.

- Client-server — Smart client code contacts the server for data, then formats and displays it to the user. Input at the client is committed back to the server when it represents a permanent change.
- 3-tier architecture — Three tier systems move the client intelligence to a middle tier so that stateless clients can be used. This simplifies application deployment. Most web applications are 3-Tier.
- N-tier architecture — N-Tier refers typically to web applications which further forward their requests to other enterprise services. This type of application is the one most responsible for the success of application servers.
- Tightly coupled (clustered) — refers typically to a set of highly integrated machines that run the same process in parallel, subdividing the task in parts that are made individually by each one, and then put back together to make the final result.
- Peer-to-peer — an architecture where there is no special machine or machines that provide a service or manage the network resources. Instead all responsibilities are uniformly divided among all machines, known as peers. Peers can serve both as clients and servers.

- Space based — refers to an infrastructure that creates the illusion (virtualization) of one single address-space. Data are transparently replicated according to application needs. Decoupling in time, space and reference is achieved.

Another basic aspect of distributed computing architecture is the method of communicating and coordinating work among concurrent processes. Through various message passing protocols, processes may communicate directly with one another, typically in a master/slave relationship. Alternatively, a "database-centric" architecture can enable distributed computing to be done without any form of direct inter-process communication, by utilizing a shared database.

1.6. CONCURRENCY

Distributed computing implements a kind of concurrency. It interrelates tightly with concurrent programming so much that they are sometimes not taught as distinct subjects.

Multiprocessor systems

A multiprocessor system is simply a computer that has >1 & not <=1 CPU on its motherboard. If the operating system is built to take advantage of this, it can run different processes (or different threads belonging to the same process) on different CPUs.

Multicore systems

Intel CPUs from the late Pentium 4 era (Northwood and Prescott cores) employed a technology called Hyperthreading that allowed more than one thread (usually two) to run on the same CPU. The more recent Sun UltraSPARC T1, AMD Athlon 64 X2, AMD Athlon FX, AMD Opteron, Intel Pentium D, Intel Core, Intel Core 2 and Intel Xeon processors feature multiple processor cores to also increase the number of concurrent threads they can run.

Multicomputer systems

A multicomputer may be considered to be either a loosely coupled NUMA computer or a tightly coupled cluster. Multicomputers are commonly used when strong compute power is required in an environment with restricted physical space or electrical power.

Common suppliers include Mercury Computer Systems, CSPI, and SKY Computers.

Common uses include 3D medical imaging devices and mobile radar.

Computing taxonomies

The types of distributed systems are based on Flynn's taxonomy of systems; single instruction, single data (SISD), single instruction, multiple data (SIMD), multiple instruction, single data (MISD), and multiple instruction, multiple data (MIMD). Other taxonomies and architectures available at Computer architecture and in Category:Computer architecture.

Computer clusters

A cluster consists of multiple stand-alone machines acting in parallel across a local high speed network. Distributed computing differs from cluster computing in that computers in a distributed computing environment are typically not exclusively running "group" tasks, whereas clustered computers are usually much more tightly coupled. Distributed computing also often consists of machines which are widely separated geographically.

Grid computing

A grid uses the resources of many separate computers, loosely connected by a network (usually the Internet), to solve large-scale computation problems. Public grids may use idle time on many thousands of computers throughout the world. Such arrangements permit handling of data that would otherwise require the power of expensive supercomputers or would have been impossible to analyze.

1.7. LANGUAGES

Nearly any programming language that has access to the full hardware of the system could handle distributed programming given enough time and code. Remote procedure calls distribute operating system commands over a network connection. Systems like CORBA, Microsoft DCOM, Java RMI and others, try to map object oriented design to the network. Loosely coupled systems communicate through intermediate documents that are typically human readable (e.g. XML, HTML, SGML, X.500, and EDI).

Languages specifically tailored for distributed programming are:

- Ada programming language
- Alef programming language
- E programming language
- Erlang programming language
- Limbo programming language
- Oz programming language
- ZPL (programming language)
- Orca programming language

1.8. LET US SUM UP

The above lesson we discussed the concept of Distributed Computing, with it organizational architecture, and also we discussed the advantages and limitations of Distributed Computing with few terms such as concurency control and various languages used to implement Distributed Computing

1.9. LESSON END ACTIVITIES

1. Explain the features of Concurency Control in Distributed Computing Envrionment

1.10. POINTS FOR DISCUSSION

1. List down various appliciton areas where distributed computing is used

1.11. REFERENCES

http://en.wikipedia.org/wiki/Image:Wikibooks-logo-en.svg

- Attiya, Hagit and Welch, Jennifer (2004). *Distributed Computing: Fundamentals, Simulations, and Advanced Topics.* Wiley-Interscience. ISBN 0471453242.
- Lynch, Nancy A (1997). *Distributed Algorithms.* Morgan Kaufmann. ISBN 1558603484.
- http://en.wikipedia.org/wiki/Distributed_computing

LESSON 2: DESIGNING OF DISTRIBUTED SYSTEMS

CONTENTS

2.0 Aim and Objectives
2.1. Introduction
2.2. Implementation Process
    2.2.1. Distributed Mutual Exclusion (DME)
    2.2.2. Centralized Approach
    2.2.3. Fully Distributed Approach
    2.2.4. Behavior of Fully Distributed Approach
2.3. Designing a Distributed Processing System
2.4. Let us Sum UP
2.5. Lesson-End Activities
2.6. Points for Discussion
2.7. References

2.0. AIM AND OBJECTIVES

At the end of this Lesson, you will be able to understand

- Distributed Mutual Exclusion (DME),
- Centralized Approach,
- Fully Distributed Approach,
- Behavior of Fully Distributed Approach, and
- Understand the Designing Process of a Distributed Systems

2.1. INTRODUCTION

The term distributed system is used to describe a system with the following characteristics: it consists of several computers that do not share a memory or a clock; the computers communicate with each other by exchanging messages over a communication network; and each computer has its own memory and runs its own operating system. The resources owned and controlled by a computer are said to be local to it, while the resources owned and controlled by other computers and those that can only be accessed through the network are said to be remote. Typically, accessing remote resources is more expensive e than accessing local resources because of the communication delays that occur in the network and the CPU overhead incurred to process communication protocols. Based on the context, the terms computer, note, host, site, machine, processor, and workstation are used interchangeably to denote a computer throughout this lesson.

2.2. IMPLEMENTATION PROCESS

Associate a timestamp with each system event

Require that for every pair of events A and B, if A $\rightarrow$ B, then the timestamp of A is less than the timestamp of B

Within each process Pi a **logical clock**, LCi is associated

The logical clock can be implemented as a simple counter that is incremented between any two successive events executed within a process

- Logical clock is **monotonically increasing**

A process advances its logical clock when it receives a message whose timestamp is greater than the current value of its logical clock

If the timestamps of two events A and B are the same, then the events are concurrent

We may use the process identity numbers to break ties and to create a total ordering

2.2.1. Distributed Mutual Exclusion (DME)

Assumptions

The system consists of *n* processes; each process *Pi* resides at a different processor

Each process has a critical section that requires mutual exclusion

Requirement

If *Pi* is executing in its critical section, then no other process *Pj* is executing in its critical section

We present two algorithms to ensure the mutual exclusion execution of processes in their critical sections

2.2.2. DME: Centralized Approach

One of the processes in the system is chosen to coordinate the entry to the critical section

A process that wants to enter its critical section sends a request message to the coordinator

The coordinator decides which process can enter the critical section next, and its sends that process a reply message

When the process receives a reply message from the coordinator, it enters its critical section

After exiting its critical section, the process sends a release message to the coordinator and proceeds with its execution

This scheme requires three messages per critical-section entry:

request

reply

release

2.2.3. DME: Fully Distributed Approach

When process *Pi* wants to enter its critical section, it generates a new timestamp, *TS*, and sends the message *request* (*Pi*, *TS*) to all other processes in the system
When process *Pj* receives a *request* message, it may reply immediately or it may defer sending a reply back
When process *Pi* receives a *reply* message from all other processes in the system, it can enter its critical section
After exiting its critical section, the process sends *reply* messages to all its deferred requests
The decision whether process *Pj* replies immediately to a *request*(*Pi, TS*) message or defers its reply is based on three factors:

If *Pj* is in its critical section, then it defers its reply to *Pi*
If *Pj* does *not* want to enter its critical section, then it sends a *reply* immediately to *Pi*
If *Pj* wants to enter its critical section but has not yet entered it, then it compares its own request timestamp with the timestamp *TS*

If its own request timestamp is greater than *TS*, then it sends a *reply* immediately to *Pi* (*Pi* asked first)
Otherwise, the reply is deferred

2.2.4. Behavior of Fully Distributed Approach

- Freedom from Deadlock is ensured
- Freedom from starvation is ensured, since entry to the critical section is scheduled according to the timestamp ordering
  - The timestamp ordering ensures that processes are served in a first-come, first served order
- The number of messages per critical-section entry is

$$2 \times (n - 1)$$

This is the minimum number of required messages per critical-section entry when processes act independently and concurrently

2.3. DESIGNING A DISTRIBUTED PROCESSING SYSTEM

In general, designing a distributed operating system is more difficult than designing a centralized operating system for several reasons.

Transparency

We saw that one of the main goals of a distributed operating system is to make the existence of multiple computers invisible (transparent) and provide a single system image to its users. That is, a distributed operating system must be designed in such a way that a collection of distinct machines connected by a communication subsystem appears to its

users as a virtual uniprocessor. Achieving complete transparency is a difficult task and requires that several different aspects of transparency be supported by the distributed operating system. The eight forms of transparency identified by the International Standards Organization's Reference Model for Open Distributed Processing [ISO 1992] are access transparency, location transparency, replication transparency, failure transparency, migration transparency, concurrency transparency, performance transparency, and scaling transparency. These transparency aspects are described below

Access Transparency

Access transparency means that users should not need or be able to recognize whether a resource (hardware or software) is remote or local. This implies that the distributed operating system should allow users to access remote resources in the same way as local resources.

Location Transparency

The two main aspects of location transparency are as follows:

1 Name transparency. This refers to the fact that the name of a resource (hardware or software) should not reveal any hint as to the physical location of the resource.

2. User mobility. This refers to the fact that no matter which machine a user is logged onto, he or she should be able to access a resource with the same name.

Replication Transparency

For better performance and reliability, almost all distributed operating systems have the provision to create replicas (additional copies) of files and other resources on different nodes of the distributed system. In these systems, both the existence of multiple copies of a replicated resource and the replication activity should be transparent to the users.

Failure Transparency

Failure transparency deals with masking from the users' partial failures in the system, such as a communication link failure, a machine failure, or a storage device crash. A distributed operating system having failure transparency property will' continue to function, perhaps in a degraded form, in the face of partial failures.

Migration Transparency

For better performance, reliability, and security reasons, an object that is capable of being moved (such as a process or a file) is often migrated from one node to another in a distributed system.

Concurrency Transparency

Concurrency transparency means that each user has a feeling that he or she is the sole user of the system and other users do not exist in the system.

Performance Transparency

The aim of performance transparency is to allow the system to be automatically reconfigured to improve performance, as loads vary dynamically in the system.

Scaling Transparency

The aim of scaling transparency is to allow the system to expand in scale without disrupting the activities of the users.

Reliability

In general, distributed systems are expected to be more reliable than centralized systems due to the existence of multiple instances of resources. However, the existence of multiple instances of the resources alone cannot increase the system's reliability. Rather, the distributed operating system, which manages these resources, must be designed properly to increase the system's reliability by taking full advantage of this characteristic feature of a distributed system.

A fault is a mechanical or algorithmic defect that may generate an error. A fault in a system causes system failure. Depending on the manner in which a failed system behaves, system failures are of two types-fail-stop.

For higher reliability, the fault-handling mechanisms of a distributed operating system must be designed properly to avoid faults, to tolerate faults, and to detect and recover from faults. Commonly used methods for dealing with these issues are fault avoidance and fault tolerance.

Flexibility

Another important issue in the design of distributed operating systems is flexibility. Flexibility is the most important feature for open distributed systems. The design of a distributed operating system should be flexible due to the following reasons:

1. Ease of modification.
2. Ease of enhancement

Performance

If a distributed system is to be used, its performance must be at least as good as a centralized system. That is, when a particular application is run on a distributed system, its overall performance should be better than or at least equal to that of running the same application on a single-processor system. However, to achieve this goal, it is important that the various components of the operating system of a distributed system be designed properly; otherwise, the overall performance of the distributed system may turn out to be worse than a centralized system. Some design principles considered useful for better performance are as follows:

1. Batch if possible.
2.Cache whenever possible.
3.Minimize copying of data.
4. Minimize network traffic.
5. Take advantage of fine-grain parallelism for multiprocessing.

Scalability

Scalability refers to the capability of a system to adapt to increased service load. It is inevitable that a distributed system will grow with time since it is very common to add new, machines or an entire sub network to the system to take care of increased workload or organizational changes in a company. Therefore, a distributed operating system should be designed to easily cope with the growth of nodes and users in the system. That is, such growth should not cause serious disruption of service or significant loss of performance to users. Some guiding principles for designing scalable distributed systems are as follows:

1. Avoid centralized entities.
2. Avoid centralized algorithms.
3. Perform most operations on client workstations.

Heterogeneity

A heterogeneous distributed system consists of interconnected sets of dissimilar hardware or software systems. Because of the diversity, designing heterogeneous distributed systems is far more difficult than designing homogeneous distributed systems in which each system is based on the same, or closely related, hardware and software. However, as a consequence of large scale, heterogeneity is often inevitable in distributed systems. Furthermore, often heterogeneity is preferred by many users because heterogeneous distributed systems provide the flexibility to their users of different computer platforms for different applications.

Security

In order that the users can trust the system and rely on it, the various resources of a computer system must be protected against destruction and unauthorized access. Enforcing security in a distributed system is more difficult than in a centralized system because of the lack of a single point of control and the use of insecure networks for data communication. Therefore, as compared to a centralized system, enforcement of security in a distributed system has the following additional requirements:

1. It should be possible for the sender of a message to know that the message was received by the intended receiver.
2. It should be possible for the receiver of a message to know that the message was sent by the genuine sender.
3. It should be possible for both the sender and receiver of a message to be guaranteed that the contents of the message were not changed while it was in transfer.

Cryptography is the only known practical method for dealing with these security aspects of a distributed system.

Emulation Of Existing Operating System

For commercial success, it is important that a newly designed distributed operating system be able to emulate existing popular operating systems such as UNIX. With this property, new software can be written using the system call interface of the new operating system to take full advantage of its special features of distribution, but a vast amount of already existing old software can also be run on the same system without the need to rewrite them. Therefore, moving to the new distributed operating system will allow both types of software to be run side by side

2.4. LET US SUM UP

The above lesson we discussed the about various implementation techniques and issues and approaches in designing a distributed processing system. Here we also discussed the role of Distributed Operating System with various concepts used in Distributed Systems.

2.5. LESSON END ACTIVITIES

1. Explain the various reasons for designing applications in Distributed Processing system

2.6. POINTS FOR DISCUSSION

1. Differentiate between Centralised approach and Fully Distributed Approach

2.7. REFERENCES http://en.wikipedia.org/wiki/Image:Wikibooks-logo-en.svg

- Attiya, Hagit and Welch, Jennifer (2004). *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Wiley-Interscience. ISBN 0471453242.
- Nadiminti, Dias de Assunção, Buyya (September 2006). "Distributed Systems and Recent Innovations: Challenges and Benefitz". *InfoNet Magazine, Volume 16, Issue 3, Melbourne, Australia*.
- http://en.wikipedia.org/wiki/Distributed_computing

UNIT – II

LESSON 3: DISTRIBUTED PROCESSING SYSTEMS

CONTENTS

3.0 Aim and Objectives
3.1. Introduction
3.2. Pros and Cons of distributed Processing
3.3. Distributed Computing System Models
3.4. Distributed Operating System
3.5. Let us Sum UP
3.6. Lesson-End Activities
3.7. Points for Discussion
3.8. References

3.0. AIM AND OBJECTIVES

At the end of this Lesson, you will be able to understand

- the advantages and Limitations of Distributed Processing,
- various types of Distributed Computing System Models,
- distributed Operating System

3.1. INTRODUCTION

The reasons behind the development of distributed systems were the availability of powerful microprocessors at low cost as well as significant advances in communication technology. The availability of powerful yet cheap microprocessors led to the development of powerful workstations that satisfy a single user's needs. These powerful stand-alone workstations satisfy user need by providing such things as bit-mapped displays and visual interfaces, which traditional time-sharing mainframe systems do not support.

When a group of people works together, there is generally a need to communicate with each other, to share data, and to share expensive resources (such as high quality printers, disk drivers, etc). This requires interconnecting computers and resources. Designing such systems became feasible with the availability of cheap and powerful microprocessors, and advances in communication technology.

When a few powerful workstations are interconnected and can communicate with each other, the total computing power available in such a system can be enormous. Such a system generally only costs tens of thousands of dollars. On the other hand, if one tries

to obtain a single machine with the computing power equal to that of a network of workstations, the cost can be as high as a few million dollars. Hence, the main advantage of distributed system is that they have a decisive price/performance advantage over more traditional time-sharing systems.

3.2. PROS AND CONS OF DISTRIBUTED PROCESSING

Resource sharing: Since a computer can request a service from another computer by sending an appropriate request to it over the communication network, hardware and software resources can be shared among computers. For example, a printer, a compiler, a text processor, or a database at a computer can be shared with remote computers.

Enhanced Performance: A distributed computing system is capable of providing rapid response time and higher system throughput. This ability is mainly due to the fact that many tasks can be concurrently executed at different computers. Moreover, distributed systems can employ a load distributing technique to improve response time. In load distributing tasks at heavily loaded computers are transferred to lightly loaded computers, thereby reducing the time tasks wait before receiving service.

Improved reliability and availability: A distributed computing system provides improved reliability and availability because a few components of the system can fail without affecting the availability of the rest of the system. Also, through the replication of data (e.g., files and directories) and services, distributed systems can be made fault tolerant. Services are processes that provide functionality (e.g., a file service provides file system management; a mail service provides an electronic mail facility).

Modular expandability: Distributed computing systems are inherently; amenable to modular expansion because new hardware and software resources can be easily added without replacing the existing resources.

3.3. DISTRIBUTED COMPUTING SYSTEM MODELS

Various models are used for building distributed computing systems. These models can be broadly classified into five categories-minicomputer, workstation, workstation-workstations server processor-pool, and hybrid. They are briefly described below

Minicomputer Model

The minicomputer model is a simple extension of the centralized time-sharing system. A distributed computing system based on this model consists of minicomputers (they may be large supercomputers as well) interconnected by a communication network. Each minicomputer usually has multiple users simultaneous logged on to it. For this, several interactive terminals are connected to each minicomputer each user is logged on to one specific minicomputer, with remote access to other Minicomputers. The network allows a user to access remote resources

that are available at some machine other than the one on to which the user is currently logged.

The minicomputer model may be used when resource sharing (such as sharing information databases of different types, with each type of database located on a different machine) with remote users is desired.

The early ARPANET is an example of a distributed computing system based minicomputer model.

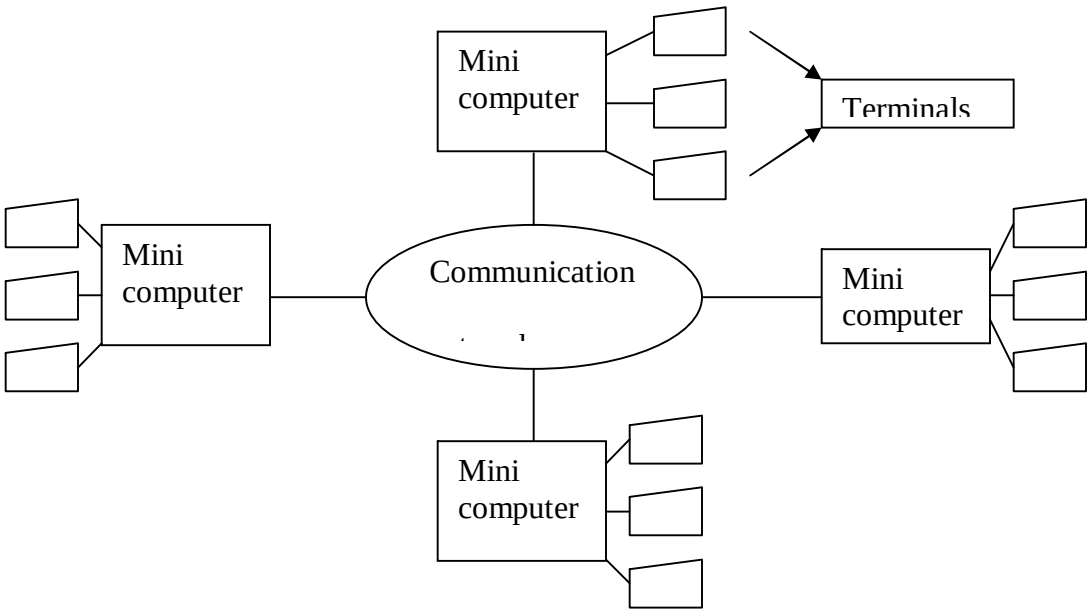


Fig 3.1 *Distributed-computing system based on the minicomputer model*

Workstation Model

The distributed computing system based on the workstation model consists of several workstations interconnected by a communication network. A company's office or a university department may have several workstations scattered throughout a building or campus, each workstation equipped with its own disk and serving as a single-user computer. It has been often found that in such an environment, at anyone time (especially at night), a significant proportion of the workstations are idle (not being used), resulting in the waste of large amounts of CPU time. Therefore, the idea of the workstation model is to interconnect all these workstations by a high-speed LAN so that idle workstations may be used to process jobs of users who are logged onto other workstations and do not have sufficient processing power at their own workstations to get their jobs processed efficiently.

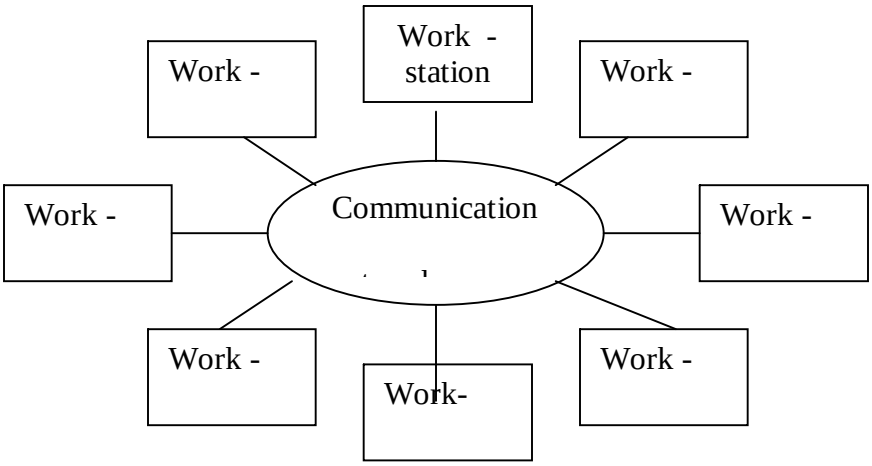


Fig. 3.2 *A distributed computing system based on the workstation model.*

Workstation – Server Model

The workstation model is a network of personal workstations, each with its own disk and a local file system. A workstation with its own local disk is usually called a diskful workstation and a workstation without a local disk is called a diskless workstation. With the proliferation of high-speed networks, diskless workstations have become more popular in network. Environments than diskful workstations, making the workstation-server model more popular than the workstation model for building distributed computing systems.

A distributed computing system based on the workstation server model consists of a few minicomputers and several workstations (most of which are diskless, but a few of which may be diskful) interconnected by a communication network.

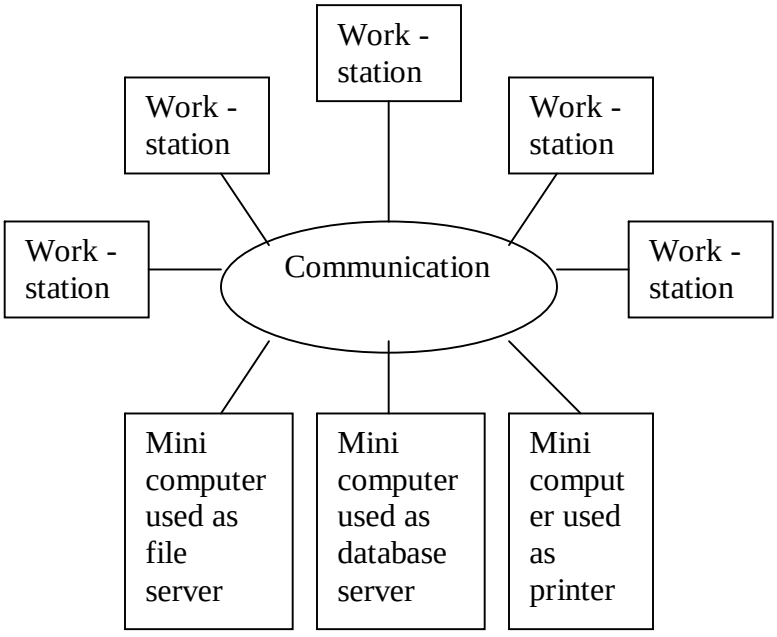


Fig 3.3. Distributed computing system based on the workstation server model.

Processor – Pool Model

The processor-pool model is based on the observation that most of the time a user does not need any computing power but once in a while he or she may need a very large amount of computing power for a short time (e.g., when recompiling a program consisting of a large number of files after changing a basic shared declaration). Therefore, unlike the workstation-server model in which a processor is allocated to each user, in the processor pool model the processors are pooled together to be shared by the users as needed. The pool of processors consists of a large number of microcomputers and minicomputers attached to the network. Each processor in the pool has its own memory to load and run a system program or an application program of the distributed computing system.

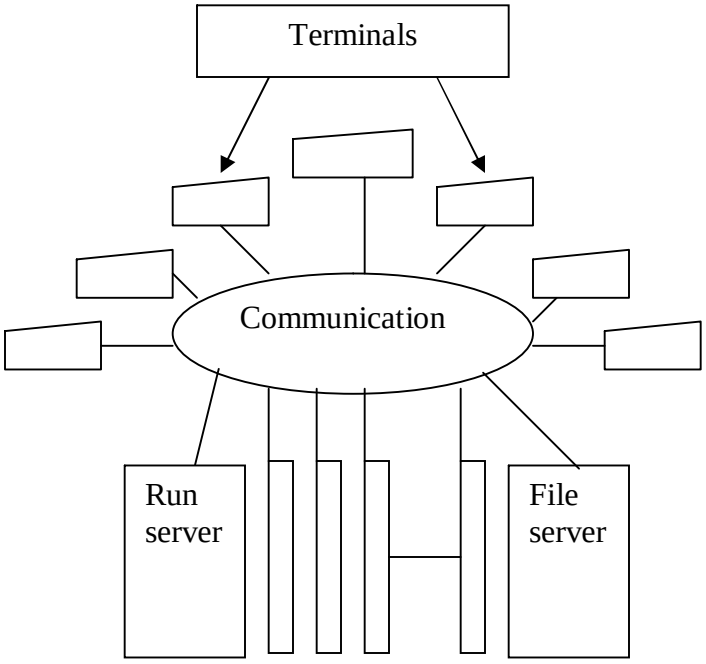


Fig 3.4. A distributed computing system based on the processor - pool model.

Hybrid Model

Out of the four models described above the workstation – server model, is the most widely used model for building distributed computing system. This is because large number of computer users only perform simple inter acting tasks such as editing jobs, sending electronic mails, and executing small programs. The workstation server model is ideal for such simple usage. However in a working environment that has groups of users who often perform jobs needing massive computation, the processor model is more attractive and suitable.

To combine the advantages of both the models a hybrid model may be used to build a distributed computing system. The hybrid model is based on the workstation server model but with the addition of pool of processors. The processors in the pool can be allocated dynamically for computations that are too large for workstations or that require several computers concurrently for efficient execution and gives a guaranteed response to interactive jobs by allowing them to be processed on local workstation of the users.

3.4 DISTRIBUTED OPERATING SYSTEM

An operating system as a program that control, the resources of a computer system and provides its users with an interface or virtual machine that is more convenient to use than the bare machine. According to this definition, the two primary tasks of an operating system are as follows

1. To present users with a virtual machine that is easier to program than the underlying hardware.
2. To manage the various resources of the system. This involves performing such tasks as keeping track of who is using which resource, granting resource requests, accounting for resource usage, and mediating conflicting requests from different programs and users

The operating systems commonly used for distributed computing systems can be broadly classified into two types-network operating systems and distributed operating systems. The three most important features commonly used to differentiate between these two types of operating systems are system image, autonomy, and fault tolerance capability. These features are explained below.

1. **System image.** The most important feature used to differentiate between the two types of operating systems is the image of the distributed computing system from the point of view of its users. In case of a network operating system, the users view the distributed computing system as a collection of distinct machines connected by a communication subsystem. A distributed operating system hides the existence of multiple computers and provides a single-system image to its users. That is, it makes a collection of networked machines act as a virtual uniprocessor.

2. **Autonomy.** In the case of a network operating system, each computer of the distributed computing system has its own local operating system (the operating systems of different computers may be the same or different), and there is essentially no coordination at all among the computers except for the rule that when two processes of different computers communicate with each other, they must use a mutually agreed on communication protocol. With a distributed operating system, there is a single system wide operating system and each computer of the distributed computing system runs a part of this global operating system. The distributed operating system tightly interweaves all the computers of the distributed computing

system in the sense that they work in close cooperation with each other for the efficient and effective utilization of the various resources of the system.

3. **Fault tolerance capability.** A network operating system provides little or no fault tolerance capability in the sense that if 10% of the machines of the entire distributed computing system are down at any moment, at least 10% of the users are unable to continue with their work. On the other hand, with a distributed operating system, most of the users are normally unaffected by the failed machines and can continue to perform their work normally, with only a 10% loss in performance of the entire distributed computing system. Therefore, the fault tolerance capability of a distributed operating system is usually very high as compared to that of a network operating system

3.5. LET US SUM UP

Distributed systems are classified into three broad categories, namely, the minicomputer model, the workstation model, and the processor pool mode.
In the minicomputer model, the distributed system consists of several minicomputers (e.g., VAXs). Each computer supports multiple users and provides access to remote resources. The ratio of the number of processors to the number of users is normally less than one.

In the workstation model, the distributed system consists of a number of workstations (up to several thousand). Each user has a workstation at his disposal, where in general, all of the user's work is performed. With the help of a distributed file system, a user can access data regardless of the location of the data or of the user's workstation. The ratio of the number of processors to the number of users is normally one. The workstations are typically equipped with a powerful processor, memory, a bit-mapped display and some cases a math co-processor and local disk storage.

In the processor pool model, the ratio of the number of processors to the number of users is normally greater than one. This model attempts to allocate one ore more microprocessors according to the user's needs. Once the processors assigned to a user complete their tasks, they return to the pool and await a new assignment. Amoeba is an experimental system that is a combination of the workstation and the processor pool models. In Amoeba, each user has a workstation where the user performs tasks that require a quick interactive response (such as editing). In addition to the workstation users have access to a pool of processors for running applications that require greater speed (such as parallel algorithms performing significant numerical computations).

3.6. LESSON END ACTIVITIES

1. Explain the various reasons for designing applications in Distributed Processing system

3.7. POINTS FOR DISCUSSION

1. Differentiate between Centralised approach and Fully Distributed Approach

3.8. REFERENCES http://en.wikipedia.org/wiki/Image:Wikibooks-logo-en.svg

- Attiya, Hagit and Welch, Jennifer (2004). *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Wiley-Interscience. ISBN 0471453242.
- Nadiminti, Dias de Assunção, Buyya (September 2006). "Distributed Systems and Recent Innovations: Challenges and Benefitz". *InfoNet Magazine, Volume 16, Issue 3, Melbourne, Australia*.
- "http://www.cs.technion.ac.il/~cs236370/main.html

LESSON 4: LOADING FACTORS

CONTENTS

4.0 Aim and Objectives
4.1. Introduction to Distributed Databases
4.2. Challenges of Distributed Data
4.3. Distributed Resource Management System
    4.3.1. Overview
    4.3.2. Main Concepts
    4.3.3. Evolution of DRMS
4.4. DRMS Responsibility
4.5.Let us Sum UP
4.6.Lesson-End Activities
4.7.Points for Discussion
4.8.References

4.0. AIM AND OBJECTIVES

At the end of this Lesson, you will be able to understand

- the need of Distributed Database,
- challenges of Distributed Data,
- distributed Resource Management System, and
- DRMS Responsibilities

4.1. INTRODUCTION TO DISTRIBUTED DATABASES

The assumption has been made up to this point that the data base under consideration is centrally managed, although of course the users might be geographically dispersed and various transactions might be executed concurrently. In actual fact many systems exist involving several different computers and a number of different databases located in a variety of different places. In such a circumstances one speaks of a distributed data base system. Possibly the best known operating distributed data base system is the world wide airline reservations system: each participating airline uses its own data base located near its particular headquarters location; common protocols are, however, used which control the operations when information involving more than one database is needed to answer a given query.

The usefulness of distributed database operations has become increasingly obvious because of the popularity of many small minicomputer systems. These systems can obviously be used to control local databases and to perform operations of interest in local environments. Furthermore, when data are need that is not locally available, the local computers can address request to a network of another machines and other databases located in various remote places.

4.2. CHALLENGES OF DISTRIBUTED DATA

A distributed data base environment complicates the systems organization and the resulting data base operations. A decision must first be made about the allocation of the files to the various locations, or nodes, of the data base network. A particular file could be kept in some unique, central place; alternatively allocating the various file portions to several different nodes could partition it. Finally, the file or certain file portion could be replicated by number of messages and request circulating from node to node assuming that the file portions of interest at a particular site are locally stored. When the data files are replicated, the message traffic between nodes may be substantially reduced. In fact, a tradeoff exists between the extra storage used by the data replication and the increased speed of operations resulting from the reduced communications load between the nodes.

In a distributed database system the need for the basic physical and logical data independence is expanded to include also location and replica transparency. Location transparency implies that the user programs are independent of the particular location of the files, while replica transparency extends the transparency to the use of an arbitrary number of copies.

Once a particular file environment is created, procedures must be available for executing the various transactions and furnishing results to the requesting parties. A given transaction might be run locally; alternatively, various remote points might be asked to carry out the operations followed by the routing of responses to the originating points. The latter strategy involves a goods deal of overhead in handling the message queues that may be formed at various points in the network.

It goes without saying that all operations must be carried out in a distributed environment in such a way that data integrity and consistency are maintained. This implies that special locking and update strategies must be used to ensure that all copies of a given database are properly updated. Specifically, all files must be locked before updating, the locks must be held until the end of a given transaction, and file alterations must be broadcast through the network to all replicas before the end of the transaction. Special transaction commit policies have been invented for this purpose in distributed systems. Specifically, a transaction coordinator is named for each transaction, and the coordinator alone is empowered to commit a given transaction after querying the participating sites concerning their individual readiness to commit.

It is obvious from what has been said that the distributed data base environment creates a host of complications. Because of the current widespread use of computer networks and the increasing availability of on-line access to computational facilities by a wide range of users, the organization and operations of distributed database systems have become popular areas of investigation for researchers in the computer field.

4.3. DISTRIBUTED RESOURCE MANAGEMENT SYSTEM

A **DRMS** is an enterprise software application that is in charge of unattended background executions, commonly known for historical reasons as batch processing.

Synonyms are **batch system**, **job scheduler**, and **Distributed Resource Manager** (DRM). Today's job schedulers typically provide a graphical user interface and a single point of control for definition and monitoring of background executions in a distributed network of computers. Increasingly job schedulers are required to orchestrate the integration of real-time business activities with traditional background IT processing, across different operating system platforms and business application environments.

4.3.1. Overview

Basic features expected of job scheduler software are:

- Interfaces to define workflows and/or job dependencies
- Automatic submission of executions
- Interfaces to monitor the executions
- Priorities and/or queues to control the execution order of unrelated jobs

If software from a completely different area includes all or some of those features, this software is considered to have job scheduling capabilities.

Most operating system platforms such as Unix and Windows provide basic job scheduling capabilities, for example Cron. Many programs such as DBMS, backup, ERPs, and BPM also include relevant job scheduling capabilities. Operating System (OS) or point program supplied job scheduling will not usually provide the ability to schedule beyond a single OS instance or outside the remit of the specific program. Organizations needing to automate highly complex related and un-related IT workload will also be expecting more advanced features from a job scheduler, such as:

- Real-time scheduling based on external, un-predictable events
- Automatic restart and recovery in event of failures
- Alerting and notification to operations personnel
- Generation of incident reports
- Audit trails for regulatory compliance purposes

These advanced capabilities can be written by in-house developers but are more often provided by solutions from suppliers that specialize in systems management software.

4.3.2. Main concepts

There are many concepts that are central to almost every job scheduler implementation and that are widely recognized with minimal variations:

- Jobs
- Dependencies
- Job Streams
- Users

Beyond the basic, single OS instance scheduling tools there are two major architectures that exist for Job Scheduling software.

- Master/Agent architecture — the historic architecture for Job scheduling software. The Job Scheduling software is installed on a single machine (Master) while on production machines only a very small component (Agent) is installed that awaits commands from the Master, executes them, and returns the exit code back to the Master.
- Cooperative architecture — a decentralized model where each machine is capable of helping with scheduling and can offload locally scheduled jobs to other cooperating machines. This enables dynamic workload balancing to maximize hardware resource utilization and high availability to ensure service delivery.

4.3.3. Evolution of DRMS

Job Scheduling has a long history. Job Schedulers are one of the major components of the IT infrastructure since the early mainframe systems. At first, stacks of punch cards were processed one after the other, hence the term “batch processing.”

From a historical point of view, we can distinguish two main eras about Job Schedulers:

1. The mainframe era
    - Job Control Language (JCL) on IBM mainframes. Initially based on JCL functionality to handle dependencies this era is typified by the development of sophisticated scheduling solutions forming part of the systems management and automation toolset on the mainframe.
2. The open systems era
    - Modern schedulers on a variety of architectures and operating systems. With standard scheduling tools limited to such as Cron, the need for mainframe standard job schedulers has grown with the increased adoption of distributed computing environments.

In terms of the type of scheduling there are also distinct eras:

1. Batch processing - the traditional date and time based execution of background tasks based on a defined period during which resources were available for batch

processing (the batch window). In effect the original mainframe approach transposed onto the open systems environment.

2. Event-driven process automation - where background processes cannot be simply run at a defined time, either because the nature of the business demands that workload is based on the occurrence of external events (such as the arrival of an order from a customer or a stock update from a store branch) or because there is no / insufficient batch window.
3. Service Oriented job scheduling - recent developments in Service Oriented Architecture (SOA) have seen a move towards deploying job scheduling as a reusable IT infrastructure service that can play a role in the integration of existing business application workload with new Web Services based real-time applications.

4.4. DRMS RESPONSIBILITY

Various schemes are used to decide which particular job to run. Parameters that might be considered include:

- Job priority
- Compute resource availability
- License key if job is using licensed software
- Execution time allocated to user
- Number of simultaneous jobs allowed for a user
- Estimated execution time
- Elapsed execution time
- Availability of peripheral devices
- Occurrence of prescribed events

4.5. LET US SUM UP

In this Lesson we discussed about Distributed Database Management System and where it is used. And we also discussed about Managing Distributed Resources and DRMS responsibility

4.6. LESSON END ACTIVITIES

1. List down various Loading factors with appropriate application example:

4.7. POINTS FOR DISCUSSION

1. Discuss various Distributed Resource Management System Funcitons

4.8. REFERENCES http://en.wikipedia.org/wiki/Image:Wikibooks-logo-en.svg

- John a . Sharp, ” An introduciton to distributed and parallel processing”, Blackwell Scientific Publicaitons.
- Nadiminti, Dias de Assunção, Buyya (September 2006). "Distributed Systems and Recent Innovations: Challenges and Benefitz”. *InfoNet Magazine, Volume 16, Issue 3, Melbourne, Australia*.
- Hewitt, Carl (April 1985). "The Challenge of Open Systems". *Byte Magazine*.

UNIT – III

LESSON 5: DATA FLOW SYSTEMS

CONTENTS

5.0 Aim and Objectives
5.1. Introduction to Distributed Databases
5.2. Communication Line Loading
5.3. Loading Calculations
5.4.Issues in Load Distributing
    5.4.1. Classification of Load Distributing Algorithms
    5.4.2. Load Balancing Vs. Load Sharing
    5.4.3. Selecting a suitable load sharing algorithm
    5.4.4. Requirement for Load Distributing
5.5. Dataflow
    5.5.1. Software architecture
    5.5.2. Hardware architecture
    5.5.3. Diagrams
    5.5.4. Concurrency
5.6. Let us Sum UP
5.7.Lesson-End Activities
5.8.Points for Discussion
5.9.References

5.0. AIM AND OBJECTIVES

At the end of this Lesson, you will be able to understand

- the various Load Balancing Calculations and about
- Data Flow Machines

5.1. INTRODUCTION

Distributed systems offer a tremendous processing capacity. However, in order to realize this tremendous computing capacity, and to take full advantage of it, good resource allocation schemes are needed. A distributed scheduler is a resource management component of a distributed operating system that focuses on judiciously and transparently redistributing the load of the system among the computers such the overall performance of the system is maximized. Because wide-are networks have high communication delays, distributed scheduling is more suitable for distributed systems based on local area networks.

In this lesson, we discuss several key issues in load distributing, including the motivation for load distributing, tradeoffs between load balancing and load sharing and between preemptive and no preemptive task transfers, and stability.

5.2. COMMUNICATION LINE LOADING

Motivation

A locally distributed system consists of a collection of autonomous computers, connected by a local area communication network. Users submit tasks at their host computers for processing. The need for load distributing arises in such environments because, due to the random arrival of tasks and their random CPU service time requirements, there is a good possibility that several computers are heavily loaded (hence suffering from performance degradation), while others are idle or lightly loaded.

Clearly, if the workload at some computers is typically heavier than that at others, or if some processors execute tasks at a slower rate than others, this situation is likely to processors are equally powerful and, over the long terms, have equally heavy workloads.

5.3. LOADING CALCULATIONS

Researches have shown that even in such homogeneous distributed systems, statistical fluctuations in the arrival of tasks and task service time requirements at computers lead to the high probability that at least one computer is idle while a task is waiting for service elsewhere. Their analysis, presented next, models a computer in a distributed system by an ***M/M/1*** server.

Consider a system of *N* identical and independent M/M/1 servers. By identical we mean that all servers have the same task arrival and service rates. Let ***a*** be the utilization of each server. Then $P_0 = 1 - \boldsymbol{a}$ is the probability that a server is idle. Let ***P*** be the probability that the system is in a state in which at least one task is waiting for service at least one server is idle. Then ***P*** is given by the expression

$$P = \sum_{i=1}^{N} \binom{N}{i} Q_i H_{N-i}$$

Where Q_i is the probability what a given set of I servers are idle and H_{N-i} is the probability that a given set of (N – i) servers are not idle and at one or more of them ***a*** task is waiting for service. Clearly, from the independence assumption,

$$\mathbf{Q_i = P_o^i}$$

H_{N-1} = {probability that N – i} systems have at least one task} – {probability that all (N-i) systems have exactly one task}

$$H_{N-i} = (1 - P_o)^{N-i} - [\{1 - P_o\} P_o]^{N-i}$$

Therefore,

$$P = \sum_{i=1}^{N} \binom{N}{i} \mathbf{P_o^i} \{(1 - P_o)^{N-i} - [\{1 - P_o\} P_o]^{N-i}\}$$

$$= \sum_{i=1}^{N} \binom{N}{i} \mathbf{P_o^i} (1 - P_o)^{N-i} - \sum_{i=1}^{N} \binom{N}{i} \mathbf{P_o^N} (1 - P_o)^{N-i}$$

$$= \{1 - (1 - \mathbf{P_o})^N\} - \{\mathbf{P_o^N} [(2 - P_o)^N (1 - P_o)^N]\}$$

$$= 1 - (1 - \mathbf{P_o})^N (1 - P_o)^N - \mathbf{P_o^N} (2 - P_o)^N$$

For moderate system utilization (where a=0.5 to -.8), the value of ***P*** is high, indicating a good potential for performance improvement through load distribution. At high system utilizations, the value of ***P*** is low as most servers are likely to be busy, which indicates lower potential for load distribution. Similarly, at low system utilizations, the value of ***P*** is low as most servers are likely to be idle, which indicates lower potential for load distribution. Another important observation is that, as the number of servers in the system increase, ***P*** remains high even at high system utilizations.

Therefore, even in a homogeneous distributed system, system performance can potentially be improved by appropriately transferring the load from heavily loaded computers (senders) to idle or lightly loaded computers (receivers). This raises the following two questions.

1. **What is mean by performance?**
   On widely used performance metric is average response time of tasks. The response time of a task is the length of the time interval between its origination and completion. Minimizing the average response time is often the goal of load distributing.

2. **What constitutes a proper characterization of load at a node?**
   Defining a proper load index is very important as load-distributing decisions are based on load measured at one or more nodes. Also, it is crucial that the mechanism used to measure load is efficient and imposes minimal overhead. These issues are discussed next.

5.4. ISSUES IN LOAD DISTRIBUTING

Here we discuss several central issues in load distributing that will help with reader understand its intricacies. Note here that the terms computer, machine, host, workstation, and node are used interchangeably, depending upon the context.

5.4.1. Classification of Load Distributing Algorithms

The basic function of a load-distributing algorithm is to transfer load (tasks) from heavily loaded computers to idle or lightly loaded computers. Load distributing algorithms can be broadly characterized as static, dynamic or adaptive. Dynamic load distributing algorithms use system state information (the loads at nodes), at least in part, to make load-distributing decisions, while static algorithms make no use of such information. In static load distributing algorithms, decisions are hard-wired in the algorithm using priori knowledge of the system. Dynamic load distributing algorithms have the potential to outperform static load distributing algorithms tributing algorithms have the potential to outperform static load distributing algorithms because they are able to exploit short term fluctuations in the system state to improve performance. However, dynamic load distributing algorithms entail overhead in the collection, storage, and analysis of system state information. Adaptive load distributing algorithms are special class of dynamic load distributing algorithms in that they adapt their activities by dynamically changing the parameters of the algorithm to suit the changing system state. For example, a dynamic algorithm may continue to collect the system state irrespective of the system load. An adaptive algorithm, on the other hand, may discontinue the collection of the system state if the overall system load is high to avoid imposing additional overhead on the system. At such loads, all nodes are likely to be busy and attempts to find receivers are unlikely to be successful.

5.4.2. Load Balancing Vs. Load Sharing

Load distributing algorithms can further be classified as load balancing or load sharing algorithms, based on their load distributing principle. Both types of algorithms strive to reduce the likelihood of an unshared state (a state in which one computer lies idle while at the same time tasks contend for service to another computer) by transferring tasks to lightly loaded nodes. Load balancing algorithms, however, go a step further by attempting to equalize loads at all computers. Because a load balancing algorithm transfers tasks to a higher rate than a load-sharing algorithm, the higher overhead incurred by the load balancing algorithm may outweigh this potential performance improvement.

Task transfers are not instantaneous because of communication delays and delays that occur during the collection of task state. Delays in transferring a task increase the duration of an unshared state, as an idle computer must wait for the arrival of the transferred task. To avoid lengthy unshared states, anticipatory task transfers from overloaded computers to computers that are likely to become idle shortly can be used. Anticipatory transfers increase that task transfer rate of a load-sharing algorithm, making it less distinguishable from load balancing algorithms. In this sense, load balancing can

be considered a special case of load sharing, performing a particular level of anticipatory task transfers.

5.4.3. Selecting a suitable load-sharing algorithm

Based on the performance trends of load sharing algorithms, one may select a load-sharing algorithm that is appropriate to the system under consideration as follows:

1. If the system under consideration never attains high loads, sender-initiated algorithms will give an improved average response time over no load sharing at all.
2. Stable scheduling algorithms are recommended for systems that can reach high loads. These algorithms perform better than non-adaptive algorithms for the following reasons:
    a. Under sender-initiated algorithms, an overloaded processor must send inquiry messages delaying the existing tasks. If an inquiry fails, two overloaded processors are adversely affected because of unnecessary message handling. Therefore, the performance impact of an inquiry is quit severe at high system loads, where most inquiries fail.
    b. Receiver-initiated algorithms remain effective at high loads but require the use of preemptive task transfers. Note that preemptive task transfers are expensive compared to non-preemptive task transfers because they involve saving and communicating a far more complicated task state.
3. For a system that experiences a wide range of load fluctuations, the stable symmetrically initiated scheduling algorithm is recommended because it provides improved performance and stability over the entire spectrum of system loads.
4. For a system that experiences wide fluctuations in load and has a high cost for the migration of partly executed tasks, stable sender-initiated algorithms are recommended, as they perform better than unstable sender-initiated algorithms at all loads, perform better than receiver-initiated algorithms over most system loads, and are stable at high loads.
5. For a system that experiences heterogeneous work arrival, adaptive stable algorithms are preferable, as they provide substantial performance improvement over non-adaptive algorithms.

5.4.4. Requirements for Load Distributing

While improving system performance is the main objective e of a load distributing scheme, there are other important requirements it must satisfy.

Scalability: It should work well in large distributed systems. This requires the ability to make quick scheduling decision with minimum overhead.

Location transparency: A distributed system should hide the location of tasks, just as a network file system hides the location of files from the user. In addition, the remote

execution of task must produce the same results it would produce if it were not transferred.

Determinism: A transferred task must produce the same results it would produce if it were not transferred.

Preemption: While utilizing idle workstations in the owner's absence improves the utilization of resources, a workstation's owner must not get a degraded performance on his return. Guaranteeing the availability of the workstation's owner must not get a degraded performance on his return. Guaranteeing the availability of the workstations' resources to its owner requires that remotely executed tasks be preempted and migrated elsewhere on demand. Alternatively, these tasks may be executed at a lower priority.

Heterogeneity: It should be able to distinguish among different architectures, processors of different processing capability, servers equipped with special hardware, etc.

5.5. DATAFLOW

Dataflow is a term used in computing, and may have various shades of meaning. It is closely related to message passing.

5.5.1. Software architecture

Dataflow is a software architecture based on the idea that changing the value of a variable should automatically force recalculation of the values of other variables.

Dataflow programming embodies these principles, with spreadsheets perhaps the most widespread embodiment of dataflow. For example, in a spreadsheet you can specify a cell formula which depends on other cells; then when any of those cells is updated the first cell's value is automatically recalculated. It's possible for one change to initiate a whole sequence of changes, if one cell depends on another cell which depends on yet another cell, and so on.

The dataflow technique is not restricted to recalculating numeric values, as done in spreadsheets. For example, dataflow can be used to redraw a picture in response to mouse movements, or to make a robot turn in response to a change in light level.

One benefit of dataflow is that it can reduce the amount of coupling-related code in a program. For example, without dataflow, if a variable X depends on a variable Y, then whenever Y is changed X must be explicitly recalculated. This means that Y is coupled to X. Since X is also coupled to Y (because X's value depends on the Y's value), the program ends up with a cyclic dependency between the two variables. Most good programmers will get rid of this cycle by using an observer pattern, but only at the cost of introducing a non-trivial amount of code. Dataflow improves this situation by making the recalculation of X automatic, thereby eliminating the coupling from Y to X. Dataflow

makes implicit a significant amount of code that otherwise would have had to be tediously explicit.

Dataflow is also sometimes referred to as reactive programming.

There have been a few programming languages created specifically to support dataflow. In particular, many (if not most) visual programming languages have been based on the idea of dataflow.

5.5.2. Hardware architecture

Hardware architectures for dataflow was a major topic in Computer architecture research in the 1970s and early 1980s. Jack Dennis of MIT pioneered the field of static dataflow architectures. Designs that use conventional memory addresses as data dependency tags are called static dataflow machines. These machines did not allow multiple instances of the same routines to be executed simultaneously because the simple tags could not differentiate between them. Designs that use Content-addressable memory are called dynamic dataflow machines by Arvind (also of MIT). They use tags in memory to facilitate parallelism.

5.5.3. Diagrams

The term dataflow may also be used to refer to the flow of data within a system, and is the name normally given to the arrows in a data flow diagram that represent the flow of data between external entities, processes, and data stores.

5.5.4. Concurrency

A dataflow network is a network of concurrently executing processes or automata that can communicate by sending data over *channels*

Kahn process networks, named after one of the pioneers of dataflow networks, are a particularly important class of such networks. In a Kahn process network the processes are *determinate*. This implies that each determinate process computes a continuous function from input streams to output streams, and that a network of determinate processes is itself determinate, thus computing a continuous function. This implies that the behaviour of such networks can be described by a set of recursive equations, which can be solved using fixpoint theory.

The concept of dataflow networks is closely related to another model of concurrency known as the Actor model.

5.5. LET US SUM UP

Load distributing algorithms try to improve the performance of distributed systems by transferring load from heavily loaded nodes to lightly loaded or idle nodes. If

task transfers are to be effective in improving the system's performance, it is important that the metric used to measure the load at nodes characterizes the load properly. The CPU queue length has been found to be a good load indicator.

In this lesson, we described several load sharing algorithms their performance and policies employed in several implementations of load distributing schemes. In addition, we discussed about the data flow systems to minimize the delay due to the transfer of state.

5.6. LESSON END ACTIVITIES

1. List down various partitioning and allocation in respect to Loading process:

54.7. POINTS FOR DISCUSSION

1. Discuss various Hardware/Software Issues in Data Flow Systems

5.8. REFERENCES http://en.wikipedia.org/wiki/Image:Wikibooks-logo-en.svg

- John a . Sharp, " An introduciton to distributed and parallel processing", Blackwell Scientific Publicaitons.
- Mukesh Singhal, Shivaratri, "Advanced Concepts in Operating System", Tata McGraw-Hill, 2001

LESSON 6: DESIGN CONSIDERATIONS

CONTENTS

6.0 Aims and Objectives
6.1. Introduction
6.2. Peer-to-Peer Networks
6.3. Client-Server Networks
6.4. Application-Server Networks
6.5. Network Database Design Considerations
6.6. Let us Sum UP
6.6. Lesson-End Activities
6.7. Points for Discussion
6.8. References

6.0. AIMS AND OBJECTIVES

After studying this Lesson, you should be able to

- describe the various Design Considerations

  Peer-to-Peer Networks
  Client-Server Networks and
  Application-Server Networks

6.1. INTRODUCTION

When accomplishing any task, there is usually more than one way to get the job done, sometimes-even thousands of different ways. When trying to connect more than one computer together there are also many different ways to get the job done. Some ways are better than others in certain situations, and it is very beneficial to get started in the right direction when networking computers, not just because of usefulness, but also because of security issues.

Just like with anything, when deciding on how to connect computers together you should investigate on why the job must be done. Question what purpose needs to be accomplished, question if any peripherals need to be shared, question if any documents need to be accessed by more than one person, etc. Once you have more information on what the network actually needs to do, then it will be much easier implementing a plan that will accomplish all of your goals.

When setting up a network there are basically three different types of networks - Peer-to-peer, Client-Server, and Application-Server Networks. Each type has certain benefits and downsides, this article will describe each type, along with why it could or

should be implemented. Those who are deciding to implement a network, or are curious about their existing network should find this article interesting and informative.

6.2. PEER-TO-PEER NETWORKS

Nearly all Operating Systems come with the ability to act as some kind of a server to share resources. You can setup different computers to allow others to use its peripherals such as printers or CDROM drives, and other computers to allow others to read or write to its hard disk allowing sharing of files, while other computers may allow access to its Internet connection. When you allow workstation computers to become servers and share things in this manner, it is called a Peer-to-peer network.

An Example of a Peer-to-peer Network

I will use a small office as an example of a Peer-to-Peer network.

In this small business office, the secretary uses the best computer, and has the most drive space, she also has a fast laser printer connected to her computer. The accountant has a mediocre computer that has a color ink jet printer. The owner has a good computer with a zip drive to take work home. All of these computers are networked together, with no central server.

The secretary uses the zip drive through the network to backup important documents, and also uses the ink jet printer on the accountant's computer to print out fliers. The accountant uses the laser printer on the secretary's computer to print out checks, accesses some important documents on the secretary's computer, and backs up the accounting data on the zip drive on the owners computer. The owner uses both printers on the other computers, and accesses important documents on the secretary's computer. All of the computers share Internet access through the secretary's computer.

All of this gets done with no passwords or user names since all the shared devices use no access control, or other type of security measure. Also in order for the accountant's computer and the owner's computer to be able to read the companies important documents, the secretary's computer must be turned on first.

The Benefits of a Peer-to-peer Network

Peer-to-peer networks are very cheap to implement because more than likely the Operating System software you have installed on your computers should have the ability to share items with other computers on the network, even though the feature may be limited. Nearly all of the most popular desktop Operating Systems have this feature, including Microsoft Windows and Apple's Mac OS, as well as Unix like OS es, such as Linux and the BSD s. So the only cost will be the networking hardware (cards, wiring, hubs or switches), and the labor to configure the workstations for this type of network sharing.

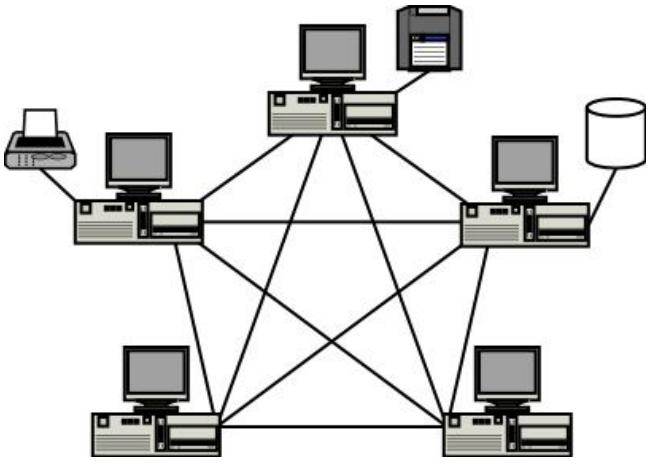
The Downsides of a Peer-to-peer Network

Even though a peer-to-peer network is very cost effective up front, there are a number of downsides you must consider before implementing this type of network.

Without a central server, it is very difficult, or nearly impossible to secure this type of network in any way. You can implement passwords on each different network share, but in order for the network to usable, the exact same username and password must be entered into each computer acting as a server. Thus, to change a password for a user could literally take hours of work, especially if the network consists of computers located in different buildings or different floors. Because of this, what usually happens with peer-to-peer networks is that passwords are implemented to begin with, but after time, either everyone starts using the exact same username and password, or the passwords end up becoming blank, or the network shares are configured to allow anyone access without a username or password. In any of these cases, security is pretty much non-existent, which can become a huge problem, especially if your network has access to the Internet.

On a peer-to-peer network, it is also very difficult to implement a good backup system because important documents tend to be stored on different hard disks on different computers. If you do manage to implement a good backup policy, chances are great that after a while some very important documents will not get archived because someone "accidentally" saved them to the wrong location on the network.

Peer-to-peer networks also tend to become very costly over time. Since each computer that shares anything to the other computers is a psuedo server, it must be constantly on, and configured correctly. So instead of maintaining a small handful of servers, you must maintain all of the workstations as servers, and as such any downtime with any computer on the network could cause considerable loss of labor or information. The following diagram illustrates all the theoretical connections that are needed for a peer-to-peer network to operate with just 5 computers. Note that this illustration does not represent physical network connections, but the theoretical network connections the operating system needs to operate the network properly.



The computer operators of a peer-to-peer network must also be well acquainted with the intricacies of running a computer in order for them to be able to do any work with the network. The users must be able to locate the different shares on the network, and be experienced enough to work through small problems, such as password problems or network mapping problems. As a side note, I have been in offices that used such complex drive mappings on a peer-to-peer network that they had a checklist showing which computers to turn in a certain order for the network to work properly.

Final Words on Peer-to-peer Networks

Peer-to-peer networks can be implemented with very little investment costs, but in order for the network to work properly, the users must be very experienced with computers, and strict guidelines must be implemented and followed in order for the data to remain secure and archived properly. In my experience, peer-to-peer networks tend to become more of a headache instead of a help after about 6 computers, especially if your company has a moderate employee turnover.

6.3. CLIENT-SERVER NETWORKS

The Client-Server network model usually consists of one or more server computers that provide services and information to a number of workstation computers. These services can consist of many different roles, including: file services, web services, email services, domain name lookup services, document version system services, Internet sharing services, etc. A great example of the Client-Server network model is actually the World Wide Internet. On the Internet clients, or computer with web browsers, access web sites that are hosted on servers.

This model differs from the Peer-to-peer network model in that the servers usually do not dually act as a workstation, and the workstations usually do not act as servers, and if they do act as a server, they should be configured to allow the central servers to provide access restrictions on the shares they provide the network.

An Example of a Client-Server Network

I will use a mid-size business with a network of 20 computers as an example.

The network is setup with a main file server, that also stores all the users email. Every night the main file server is backed up to a secondary file server that is located in an adjacent building. The network also has a network firewall computer that serves an Internet connection to the network, and forwards all email to the file server. The file server has different shares for each department so only people in that department has access to the files, and also has a company wide share that everyone in the company has access to the files. Each user also has a home directory on the main file server for personal documents that no one else can access. The network also has two large laser

printers and a commercial color laser printer, all the printers are connected to the main file server through a jet direct interface.

The client computers all map a drive letter (L:) to the company wide share, and also map a drive letter (M:) to the department share, and the clients can print to whichever printer they need to. The drive mappings occur during a login script that is ran when the client computer logs in to the network. Each department has at least one zip drive on its computers which is shared for backups and convenience. Each employee has a separate email account, and all email messages reside on the main file server, so if any of the workstation computers goes down, the emails are still intact.

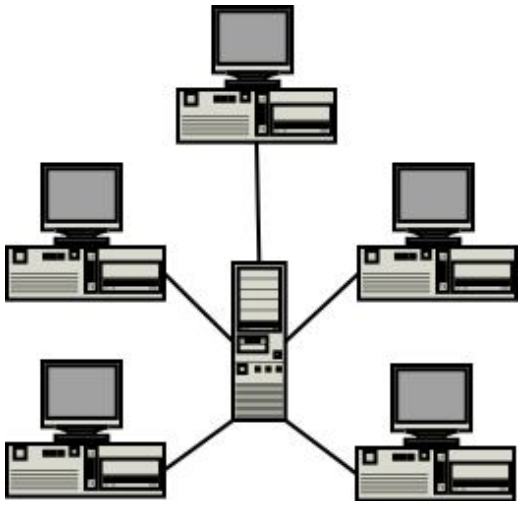
The Benefits of a Client-Server Network

The Client-Server network model offers many benefits that stems from the fact that only a single computer, or group of computers are the central repository for all the networking services that you may need.

Security is relatively easy to implement with this type of network model, since you can setup a single server computer to handle all information requests or login requests for the entire network, thus you only need one username and password for each user on the network. So if you ever need to change a password, you only need to change it at the server and the password would be changed for the entire network.

Information control is also fundamentally easier with this type of network model because you can have individual server computers store all the important documents of you company on a single store. In doing this you gain the ability to easily archive all the companies' documents, as well as provide a secure, easy to access network store for all of your users, reducing the possibility of misplaced documents on your network. Other information can also be controlled by individual servers, such as all of the company's email and contact lists can be stored on a single mail server, or all of the company's policies and public documents can be store on an internal web server or ftp server.

With the Client-Server network model, each workstation only really needs to have one theoretical connection on the network, and that connection is to the main server as illustrated in the image below. Because of this, the maintenance cost for the network drops. Also, since all the important information of the network actually resides on the servers, the workstation maintenance also drops since the users can access any information they need through any workstation, and a faulty workstation computer will have very little effect on the usefulness of the network. I actually have setup networks where the workstation computers are backed up to an image on a central server, so if a workstation goes down, a technician can restore the image and have the workstation back up literally within minutes.



There are numerous other benefits to this type of network, most stem from the fact that you consolidate information or security to a single computer, or groups of computers. Once this is done, adding other services to the network is both easier and more secure.

The Downsides of a Client-Server Network

Even though the Client-Server type of network has many advantages, there are some disadvantages that you should be aware of.

The cost of this type of network is relatively high up front, not only must you purchase the server hardware, but most server software is very expensive, especially for larger networks since some software companies charge more for each client computer that will connect to the main server (although there are cheaper alternatives). Once the network is in place however, it is relatively easy to justify the cost since the overall cost to maintain the network becomes less expensive.

Another downside to consider is the possibility of the main server having problems. How fast must you have the network working again? If you need 24x7 operability, you should allow in your budget a second "redundant" server, so if the main server goes down, the redundant server will step in and provide services until the primary server is back up again. An experienced administrator should be able to setup redundant servers that will assume control of failing servers without user intervention.

Final Words on Client-Server Networks

The Client-Server network model provides important services to the network safely and securely, it also allows the convenience of allowing the users to work on their own workstation machine. However, this network model can be very expensive, not only

because the software can be expensive, but you also must provide adequate hardware for both the servers and the individual workstation machines, which can become very expensive with revolving hardware updates.

If you have the funds to implement this type of network, the return on the investment is great, and you will have the knowledge that your network is well secured and archived.

6.4. APPLICATION-SERVER NETWORKS

Overview of Application-Server Networks

The final network type that I am going to cover is the Application Server based Networks, sometimes called Terminal Server based. The idea behind this type of network is that you basically have one high-end server or mainframe, and all the network clients are "dumb terminals", meaning that none of the processing is actually done on the terminals, instead the only job the terminals have is to provide input and show the display on the monitor.

Most people equate application servers to the very old text-only terminals with no pointing devices. Today application servers are very modern, and most people running on a "dumb terminal" will think they are working on a modern standalone computer.

An Example of an Application-Server Network

I will use a Metropolitan Library located in a three story building, with 20 terminal computers on each floor as an example.

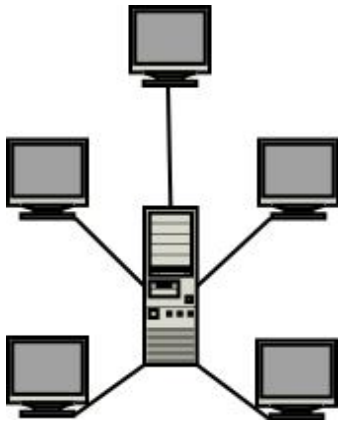
Each floor has its own Application Server running a version of Linux. Each application server has basic user applications, such as Internet Browser, Word Processor, Spreadsheet Program, Email Application, Image Manipulation Program, as well as all the basic applications you should find on a computer. Each Application Server serves applications to 20 different terminals, which are older donated computers. Each terminal has the ability to run all of the above applications, print to any of the printers on each floor and has access to the main card catalog through a web-based interface.

If one of the Application Servers goes down, the network is configured so that the terminals will log into one of the other floors servers until the computer is repaired. If a terminal goes down, a replacement terminal can be installed with no downtime for the entire network, with no information loss.

All the Application Servers also share a single /home directory from a separate File Server, which allows the library the ability to offer an individual login name, email account and individual storage for a small charge. Along with the login name, email account and storage, the patron also has the ability to access any files he may have saved on the file server through a secure FTP server.

Benefits of Application-Server Networks

The biggest benefit that this type of network provides is cost. It is very cheap to implement and maintain an Application Server based network. The only high end component you need is a high quality server computer with lots and lots of memory. As for the terminals, they can be purchased very cheaply, or one could even use old 486 and Pentium computers and not notice any slowdown.



The maintenance of this type of network is also very low cost, since you basically only need to maintain the one or two servers that provide the applications. Also, to lower the cost even more, you can install and use commodity software, such as Linux or BSD Unix, which can be obtained with little or no cost.

The Downsides of Application-Server Networks

The downside to running all of the clients on one server is, of course, what happens when the server goes down. This of course is a huge disadvantage, but one that can be overcome with installing a second or even third Application Server to the network. Which would also spread out the connections across the servers, so that the performance would not diminish as much when more and more users access the servers.

Another downside is the fact that most Proprietary Software packages are licensed, and most will not allow you to run the software on Application Servers without a substantial monetary investment. You can combat this cost by sticking with Open Source variants of commodity software, such as Word Processors, Web Browsers and Email Applications, and use standalone computers for the specialized software such as accounting software.

Final Words on Application-Server Networks

Even though not every software package will allow you to run it off of an Application Server, the price benefits can be astounding when this type of network is Implemented. If you need to provide public access to computers, or have separate departments that only need to use word processing, spreadsheets, and email, an Application Server could literally save you tens of thousands of dollars, even on a smaller network of 10-20 computers.

6.5. NETWORK DATABASE DESIGN CONSIDERATIONS

When designing a database, it is important to consider which tables users should be able to access. Access to tables is granted or revoked through authorizations. The highest level of authority is system administration authority (SYSADM). A user with SYSADM authority can assign other authorizations, including database administrator authority (DBADM).

For *audit* purposes, you may have to record every update made to your data for a specified period. For example, you may want to update an audit table each time an employee's salary is changed. Updates to this table could be made automatically if an appropriate trigger is defined. Audit activities can also be carried out through the DB2(R) Universal Database (DB2 UDB) audit facility.

For performance reasons, you may only want to access a selected amount of data, while maintaining the base data as *history*. You should include within your design, the requirements for maintaining this historical data, such as the number of months or years of data that is required to be available before it can be purged.

You may also want to make use of *summary* information. For example, you may have a table that has all of your employee information in it. However, you would like to have this information divided into separate tables by division or department. In this case, a materialized query table for each division or department based on the data in the original table would be helpful.

Security implications should also be identified within your design. For example, you may decide to support user access to certain types of data through security tables. You can define access levels to various types of data, and who can access this data. Confidential data, such as employee and payroll data would have stringent security restrictions.

You can create tables that have a *structured type* associated with them. With such typed tables, you can establish a hierarchical structure with a defined relationship between those tables called a *type hierarchy*. The type hierarchy is made up of a single root type, supertypes, and subtypes.

A *reference type* representation is defined when the root type of a type hierarchy is created. The target of a reference is always a row in a typed table or view.

When working in a High Availability Disaster Recovery (HADR) environment, there are several recommendations:

- The two instances of the HADR environment should be identical in hardware and software. This means:
    - The host computers for the HADR primary and standby databases should be identical.
    - The operating system on the primary and standby systems should have the same version including patches. (During an upgrade this may not be possible. However, the period when the instances are out of step should be kept as short as possible to limit any difficulties arising from the differences. Those differences could include the loss of support for new features during a failover as well as any issues affecting normal non-failover operations.)
    - A TCP/IP interface must be available between the two HADR instances.
    - A high speed, high capacity network is recommended.
- There are DB2 UDB requirements that should be considered.
    - The database release used by the primary and the standby should be identical.
    - The primary and standby DB2 UDB software must have the same bit size. (That is, both should be at 32-bit or at 64-bit.)
    - The table spaces and their corresponding containers should be identical on the primary and standby databases. Those characteristics that must be symmetrical for the table spaces include (but are not limited to): the table space type (DMS or SMS), table space size, the container paths, the container sizes, and the container file type (raw device or file system). Relative container paths may be used, in which case the relative path must be the same on each instance; they may map to the same or different absolute paths.
    - The primary and standby databases need not have the same database path (as declared when using the CREATE DATABASE command).
    - Buffer pool operations on the primary are replayed on the standby, which suggests the importance of the primary and standby databases having the same amount of memory.

Updating a single database in a transaction

The simplest form of transaction is to read from and write to only one database within a single unit of work. This type of database access is called a *remote unit of work*.

Figure . Using a single database in a transaction

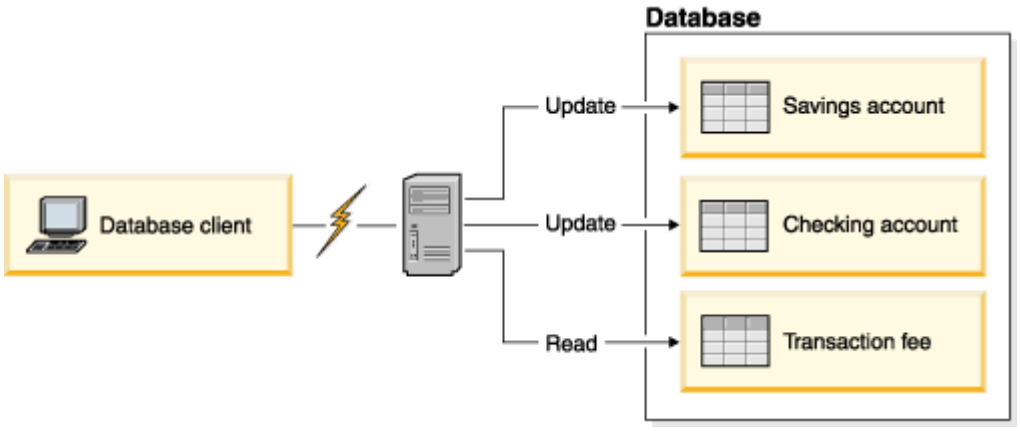


Figure shows a database client running a funds transfer application that accesses a database containing checking and savings account tables, as well as a banking fee schedule. The application must:

- Accept the amount to transfer from the user interface
- Subtract the amount from the savings account, and determine the new balance
- Read the fee schedule to determine the transaction fee for a savings account with the given balance
- Subtract the transaction fee from the savings account
- Add the amount of the transfer to the checking account
- Commit the transaction (unit of work).

Procedure

To set up such an application, you must:

1. Create the tables for the savings account, checking account and banking fee schedule in the same database
2. If physically remote, set up the database server to use the appropriate communications protocol
3. If physically remote, catalog the node and the database to identify the database on the database server
4. Precompile your application program to specify a type 1 connection; that is, specify CONNECT 1 (the default) on the PRECOMPILE PROGRAM command.

Updating a single database in a multi-database transaction

If your data is distributed across multiple databases, you may wish to update one database while reading from one or more other databases. This type of access can be performed within a single unit of work (transaction).

Figure Using multiple databases in a single transaction

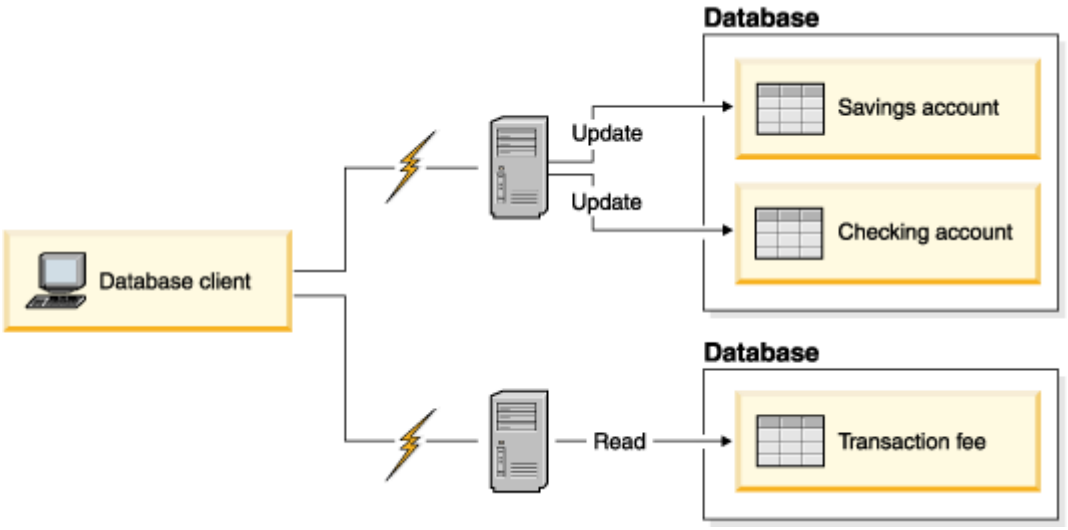


Figure shows a database client running a funds transfer application that accesses two database servers: one containing the checking and savings accounts, and another containing the banking fee schedule.

Procedure

To set up a funds transfer application for this environment, you must:

1. Create the necessary tables in the appropriate databases
2. If physically remote, set up the database servers to use the appropriate communications protocols
3. If physically remote, catalog the nodes and the databases to identify the databases on the database servers
4. Precompile your application program to specify a type 2 connection (that is, specify CONNECT 2 on the PRECOMPILE PROGRAM command), and one-phase commit (that is, specify SYNCPOINT ONEPHASE on the PRECOMPILE PROGRAM command).

If databases are located on a host or iSeries database server, you require DB2 Connect(TM) for connectivity to these servers.

Updating multiple databases in a transaction

If your data is distributed across multiple databases, you may want to read and update several databases in a single transaction. This type of database access is called a *multisite update*.

Figure. Updating multiple databases in a single transaction

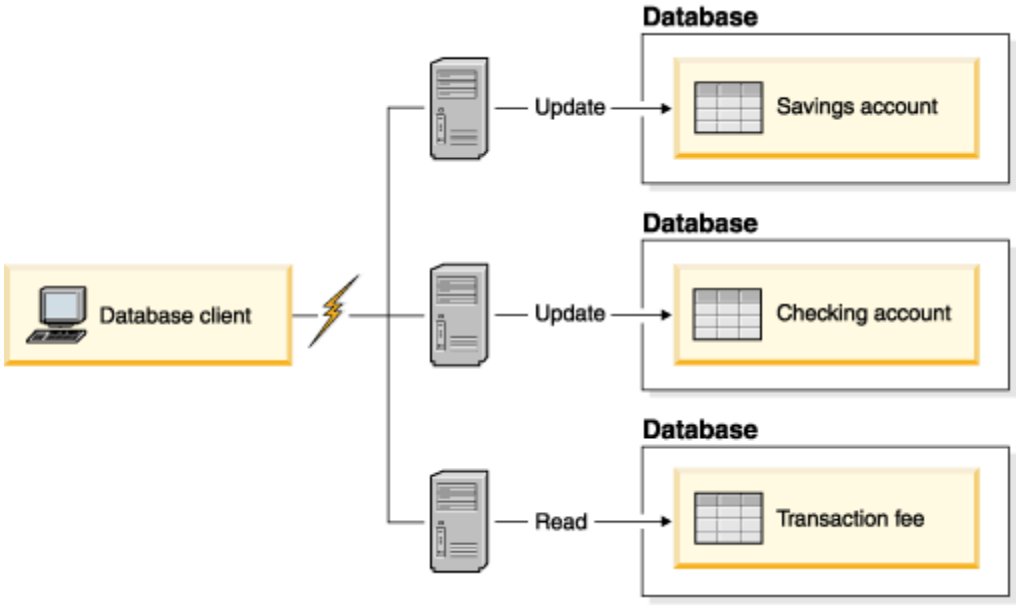


Figure shows a database client running a funds transfer application that accesses three database servers: one containing the checking account, another containing the savings account, and the third containing the banking fee schedule.

Procedure

To set up a funds transfer application for this environment, you have two options:

2. With the DB2 Universal Database(TM) (DB2 UDB) transaction manager (TM):
    a. Create the necessary tables in the appropriate databases
    b. If physically remote, set up the database servers to use the appropriate communications protocols
    c. If physically remote, catalog the nodes and the databases to identify the databases on the database servers
    d. Precompile your application program to specify a type 2 connection (that is, specify CONNECT 2 on the PRECOMPILE PROGRAM command), and two-phase commit (that is, specify SYNCPOINT TWOPHASE on the PRECOMPILE PROGRAM command)
    e. Configure the DB2 UDB transaction manager (TM).
3. Using an XA-compliant transaction manager:
    a. Create the necessary tables in the appropriate databases
    b. If physically remote, set up the database servers to use the appropriate communications protocols
    c. If physically remote, catalog the nodes and the databases to identify the databases on the database servers
    d. Precompile your application program to specify a type 2 connection (that is, specify CONNECT 2 on the PRECOMPILE PROGRAM command), and one-phase commit (that is, specify SYNCPOINT ONEPHASE on the PRECOMPILE PROGRAM command)

e. Configure the XA-compliant transaction manager to use the DB2 UDB databases.

DB2 transaction manager

- The DB2(R) Universal Database (DB2 UDB) transaction manager (TM) assigns identifiers to transactions, monitors their progress, and takes responsibility for transaction completion and failure. DB2 UDB and DB2 Connect(TM) provide a transaction manager. The DB2 UDB TM stores transaction information in the designated TM database.
- The database manager provides transaction manager functions that can be used to coordinate the updating of several databases within a single unit of work. The database client automatically coordinates the unit of work, and uses a *transaction manager database* to register each transaction and track its completion status.
- You can use the DB2 UDB transaction manager with DB2 UDB databases. If you have resources other than DB2 UDB databases that you want to participate in a two-phase commit transaction, you can use an XA-compliant transaction manager.

DB2 Universal Database transaction manager configuration

If you are using an XA-compliant transaction manager, such as IBM(R) WebSphere(R), BEA Tuxedo, or Microsoft(R) Transaction Server, you should follow the configuration instructions for that product.

When using DB2(R) Universal Database (DB2 UDB) for UNIX(R)-based systems or the Windows(R) operating system to coordinate your transactions, you must fulfill certain configuration requirements. If you use TCP/IP exclusively for communications, and DB2 UDB for UNIX, Windows, iSeries(TM) V5, z/OS(TM) or OS/390(R) are the only database servers involved in your transactions, configuration is straightforward.

DB2 Connect(TM) no longer supports SNA two phase commit access to host or iSeries servers.

DB2 Universal Database for UNIX and Windows and DB2 for z/OS, OS/390, and iSeries V5 using TCP/IP Connectivity

If each of the following statements is true for your environment, the configuration steps for multisite update are straightforward.

- All communications with remote database servers (including DB2 UDB for z/OS, OS/390, and iSeries V5) use TCP/IP exclusively.
- DB2 UDB for UNIX based systems, Windows operating systems, z/OS, OS/390 or iSeries V5 are the only database servers involved in the transaction.

The database that will be used as the transaction manager database is determined at the database client by the database manager configuration parameter *tm_database*. Consider the following factors when setting this configuration parameter:

- The transaction manager database can be:
  - A DB2 UDB for UNIX or Windows Version 8 database
  - A DB2 for z/OS and OS/390 Version database or a DB2 for OS/390 Version 5 or 6 database
  - A DB2 for iSeries V5 database

    DB2 for z/OS, OS/390, and iSeries V5 are the recommended database servers to use as the transaction manager database. z/OS, OS/390, and iSeries V5 systems are, generally, more secure than workstation servers, reducing the possibility of accidental power downs, reboots, and so on. Therefore the recovery logs, used in the event of resynchronization, are more secure.

- If a value of 1ST_CONN is specified for the *tm_database* configuration parameter, the first database to which an application connects is used as the transaction manager database.

  Care must be taken when using 1ST_CONN. You should only use this configuration if it is easy to ensure that all involved databases are cataloged correctly; that is, if the database client initiating the transaction is in the same instance that contains the participating databases, including the transaction manager database.

Note that if your application attempts to disconnect from the database being used as the transaction manager database, you will receive a warning message, and the connection will be held until the unit of work is committed.

Configuration parameters

You should consider the following configuration parameters when you are setting up your environment.

Database Manager Configuration Parameters

- *tm_database*

  This parameter identifies the name of the Transaction Manager (TM) database for each DB2 UDB instance.

- *spm_name*

  This parameter identifies the name of the DB2 Connect sync point manager instance to the database manager. For resynchronization to be successful, the name must be unique across your network.

- *resync_interval*

  This parameter identifies the time interval (in seconds) after which the DB2 Transaction Manager, the DB2 UDB server database manager, and the DB2 Connect sync point manager or the DB2 UDB sync point manager should retry the recovery of any outstanding indoubt transactions.

- *spm_log_file_sz*

  This parameter specifies the size (in 4 KB pages) of the SPM log file.

- *spm_max_resync*

  This parameter identifies the number of agents that can simultaneously perform resynchronization operations.

- *spm_log_path*

  This parameter identifies the log path for the SPM log files.

Database Configuration Parameters

- *maxappls*

  This parameter specifies the maximum permitted number of active applications. Its value must be equal to or greater than the sum of the connected applications, plus the number of these applications that may be concurrently in the process of completing a two-phase commit or rollback, plus the anticipated number of indoubt transactions that might exist at any one time.

- *autorestart*

  This database configuration parameter specifies whether the RESTART DATABASE routine will be invoked automatically when needed. The default value is YES (that is, enabled).

  A database containing indoubt transactions requires a restart database operation to start up. If *autorestart* is not enabled when the last connection to the database is dropped, the next connection will fail and require an explicit RESTART DATABASE invocation. This condition will exist until the indoubt transactions

have been removed, either by the transaction manager's resynchronization operation, or through a heuristic operation initiated by the administrator. When the RESTART DATABASE command is issued, a message is returned if there are any indoubt transactions in the database. The administrator can then use the LIST INDOUBT TRANSACTIONS command and other Command Line Processor (CLP) commands to find get information about those indoubt transactions.

Updating a database from a host or iSeries client

Applications executing on host or iSeries can access data residing on DB2 Universal Database(TM) (DB2 UDB) database servers. TCP/IP is the only protocol used for this access. DB2 UDB servers on all platforms no longer support SNA access from remote clients.

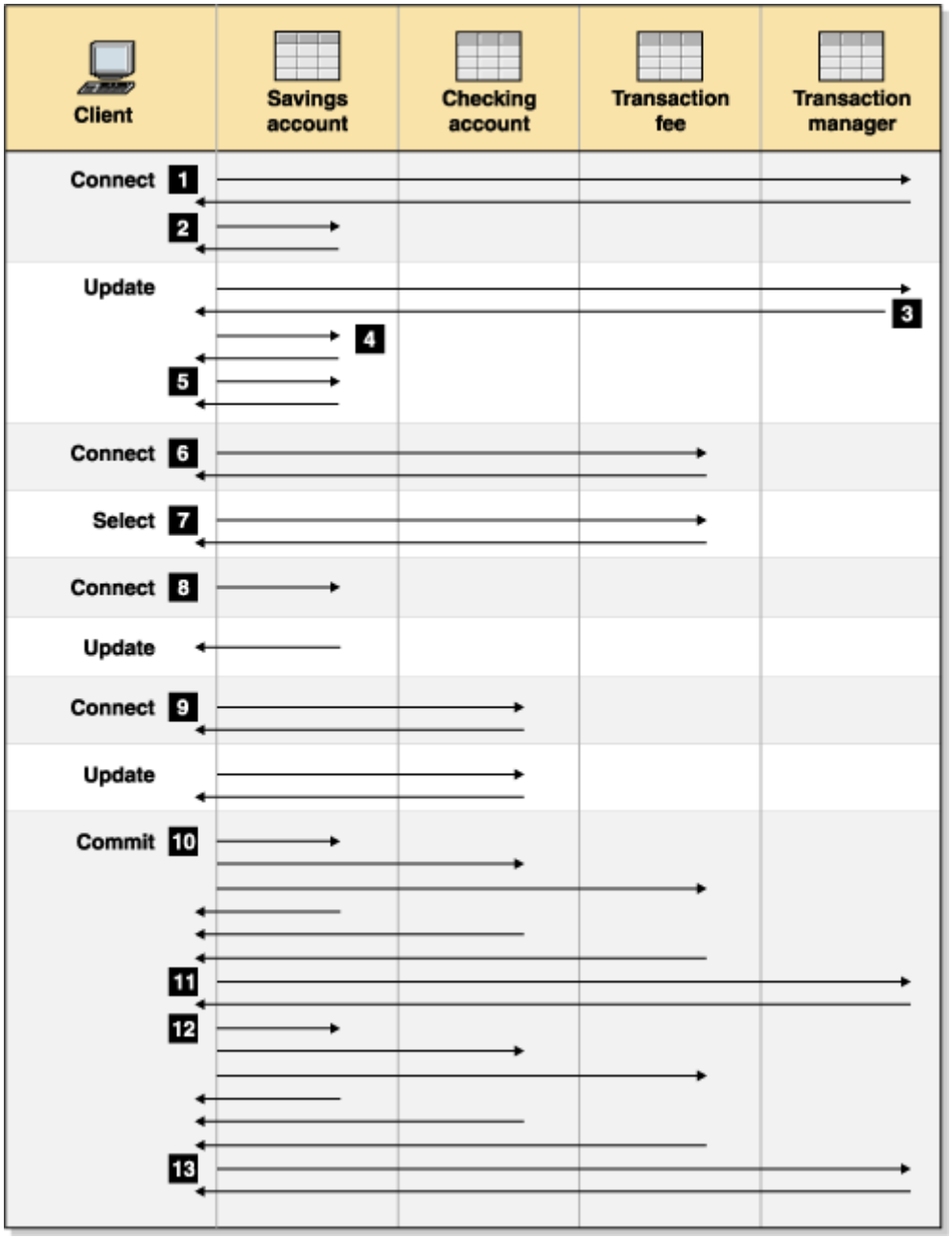
Previous to version 8, TCP/IP access from host or iSeries clients only supported one-phase commit access. DB2 UDB now allows TCP/IP two-phase commit access from host or iSeries clients. There is no need to use the Syncpoint Manager (SPM) when using TCP/IP two-phase commit access.

The DB2 UDB TCP/IP listener must be active on the server to be accessed by the host or iSeries client. You can check that the TCP/IP listener is active by using the db2set command to validate that the registry variable DB2COMM has a value of "tcpip"; and that the database manager configuration parameter **svcename** is set to the service name by using the GET DBM CFG command. If the listener is not active, it can be made active by using the db2set command and the UPDATE DBM CFG command.

Two-phase commit

Figure illustrates the steps involved in a multisite update. Understanding how a transaction is managed will help you to resolve the problem if an error occurs during the two-phase commit process.

Figure Updating multiple databases



0 The application is prepared for two-phase commit. This can be accomplished through precompilation options. This can also be accomplished through DB2(R) Universal Database (DB2 UDB) CLI (Call Level Interface) configuration.

1 When the database client wants to connect to the SAVINGS_DB database, it first internally connects to the transaction manager (TM) database. The TM database returns an acknowledgment to the database client. If the database manager configuration parameter *tm_database* is set to 1ST_CONN, SAVINGS_DB becomes the transaction manager database for the duration of this application instance.

2 The connection to the SAVINGS_DB database takes place and is acknowledged.

3 The database client begins the update to the SAVINGS_ACCOUNT table. This begins the unit of work. The TM database responds to the database client, providing a transaction ID for the unit of work. Note that the registration of a unit of work occurs when the first SQL statement in the unit of work is run, not during the establishment of a connection.

4 After receiving the transaction ID, the database client registers the unit of work with the database containing the SAVINGS_ACCOUNT table. A response is sent back to the client to indicate that the unit of work has been registered successfully.

5 SQL statements issued against the SAVINGS_DB database are handled in the normal manner. The response to each statement is returned in the SQLCA when working with SQL statements embedded in a program.

6 The transaction ID is registered at the FEE_DB database containing the TRANSACTION_FEE table, during the first access to that database within the unit of work.

7 Any SQL statements against the FEE_DB database are handled in the normal way.

8 Additional SQL statements can be run against the SAVINGS_DB database by setting the connection, as appropriate. Since the unit of work has already been registered with the SAVINGS_DB database **4**, the database client does not need to perform the registration step again.

9 Connecting to, and using the CHECKING_DB database follows the same rules described in **6** and .

10 When the database client requests that the unit of work be committed, a *prepare* message is sent to all databases participating in the unit of work. Each database writes a "PREPARED" record to its log files, and replies to the database client.

11 After the database client receives a positive response from all of the databases, it sends a message to the transaction manager database, informing it that the unit of work is now ready to be committed (PREPARED). The transaction manager database writes a "PREPARED" record to its log file, and sends a reply to inform the client that the second phase of the commit process can be started.

12 During the second phase of the commit process, the database client sends a message to all participating databases to tell them to commit. Each database writes a "COMMITTED" record to its log file, and releases the locks that were held for this unit of work. When the database has completed committing the changes, it sends a reply to the client.

13 After the database client receives a positive response from all participating databases, it sends a message to the transaction manager database, informing it that the unit of work has been completed. The transaction manager database then writes a "COMMITTED" record to its log file, indicating that the unit of work is complete, and replies to the client, indicating that it has finished.

Error recovery during two-phase commit

Recovering from error conditions is a normal task associated with application programming, system administration, database administration and system operation. Distributing databases over several remote servers increases the potential for error resulting from network or communications failures. To ensure data integrity, the database manager provides the two-phase commit process. The following explains how the database manager handles errors during the two-phase commit process:

- **First Phase Error**

  If a database communicates that it has failed to prepare to commit the unit of work, the database client will roll back the unit of work during the second phase of the commit process. A prepare message will *not* be sent to the transaction manager database in this case.

  During the second phase, the client sends a rollback message to all participating databases that successfully prepared to commit during the first phase. Each database then writes an "ABORT" record to its log file, and releases the locks that were held for this unit of work.

- **Second Phase Error**

  Error handling at this stage is dependent upon whether the second phase will commit or roll back the transaction. The second phase will only roll back the transaction if the first phase encountered an error.

  If one of the participating databases fails to commit the unit of work (possibly due to a communications failure), the transaction manager database will retry the commit on the failed database. The application, however, will be informed that the commit was successful through the SQLCA. DB2(R) Universal Database (DB2 UDB) will ensure that the uncommitted transaction in the database server is committed. The database manager configuration parameter *resync_interval* is used to specify how long the transaction manager database should wait between attempts to commit the unit of work. All locks are held at the database server until the unit of work is committed.

  If the transaction manager database fails, it will resynchronize the unit of work when it is restarted. The resynchronization process will attempt to complete all *indoubt transactions*; that is, those transactions that have finished the first phase, but have not completed the second phase of the commit process. The database manager associated with the transaction manager database performs the resynchronization by:

  1. Connecting to the databases that indicated they were "PREPARED" to commit during the first phase of the commit process.

2. Attempting to commit the indoubt transactions at those databases. (If the indoubt transactions cannot be found, the database manager assumes that the database successfully committed the transactions during the second phase of the commit process.)
3. Committing the indoubt transactions in the transaction manager database, after all indoubt transactions have been committed in the participating databases.

If one of the participating databases fails and is restarted, the database manager for this database will query the transaction manager database for the status of this transaction, to determine whether the transaction should be rolled back. If the transaction is not found in the log, the database manager assumes that the transaction was rolled back, and will roll back the indoubt transaction in this database. Otherwise, the database waits for a commit request from the transaction manager database.

If the transaction was coordinated by a transaction processing monitor (XA-compliant transaction manager), the database will always depend on the TP monitor to initiate the resynchronization.

If, for some reason, you cannot wait for the transaction manager to automatically resolve indoubt transactions, there are actions you can take to manually resolve them. This manual process is sometimes referred to as "making a heuristic decision".

Error recovery if autorestart=off

If the *autorestart* database configuration parameter is set to OFF, and there are indoubt transactions in either the TM or RM databases, the RESTART DATABASE command is required to start the resynchronization process. When issuing the RESTART DATABASE command from the command line processor, use different sessions. If you restart a different database from the same session, the connection established by the previous invocation will be dropped, and must be restarted once again. Issue the TERMINATE command to drop the connection after no more indoubt transactions are returned by the LIST INDOUBT TRANSACTIONS command.

Rule and Decision Tree Induction

Rules are the probably the most common form of representation. They tend to be simple and intuitive, unstructured and less rigid. As the drawbacks they are difficult to maintain, and they are inadequate to represent many types of knowledge.

A database is a store of information but more important is the information which can be inferred from it. There are two main inference techniques available ie deduction and induction.

- *Deduction* is a technique to infer information that is a logical consequence of the information in the database e.g. the join operator applied to two relational tables where the first concerns employees and departments and the second departments and managers infers a relation between employee and managers.
- *Induction* has been described earlier as the technique to infer information that is generalised from the database as in the example mentioned above to infer that each employee has a manager. This is higher level information or knowledge in that it is a general statement about objects in the database. The database is searched for patterns or regularities.

Induction has been used in the following ways within data mining.

- *Creation of decision trees.* Decision trees are simple knowledge representation and they classify examples to a finite number of classes, the nodes are labelled with attribute names, the edges are labelled with possible values for this attribute and the leaves labelled with different classes. Objects are classified by following a path down the tree, by taking the edges, corresponding to the values of the attributes in an object.
- *Rule induction.* A data mine system has to infer a model from the database that is it may define classes such that the database contains one or more attributes that denote the class of a tuple, i.e., the predicted attributes while the remaining attributes are the predicting attributes. Class can then be defined by condition on the attributes. When the classes are defined the system should be able to infer the rules that govern classification, in other words the system should find the description of each class.
- The relationship between Data Mining (DM) and Machine Learning (ML) may also be considered. Data mining is about finding understandable knowledge. On the other hand, machine learning is concerned with improving performance of a system. For example, training a neural network to balance a pole is part of ML, but not of KDD. Efficiency of the algorithm and scalability is more important in data mining. DM is concerned with very large, real-world databases. The techniques used in DM have often their origin in ML. For an overview on machine learning techniques

Synchronization of network databases

A method of synchronization applied to databases of a network management system which establishes synchronization between a database of a managing system and a database of a managed system, and thereby facilitates the establishment of an initial database and reconfiguration of the same. They are:

1. establishes a database based on a management information tree modeled on the system configuration for a network element NE forming a managed system,
2. transfers an upper layer managed object instance information of the management information tree to an operation system OS of the managing system autonomously or by issuance of a command,
3. starts, by the operation system OS of the managing system, the establishment of the database by the upper layer managed object instance data,
4. demands, by this operation system OS, information of a lower layer managed object instance subordinate to the upper layer managed object instance,

5. sends, by the network element NE of the managed system, the lower layer managed object instance information to the managing system, and
6. establish, by the managing system, the database using the thus sent information.

What is claimed is:

1. A method of synchronization applied to databases in a network management system comprising a managing system and at least one managed system managed by the managing system, comprising the steps of:
    - setting up, by said managed system, a database of a management information tree configuration modeled on the system configuration;
    - transferring, from said managed system to said managing system, information of an upper layer hierarchy in said management information tree configuration so that said managing system may set up its own database;
    - starting, by said managing system, a set up of its database based on the information of said upper layer hierarchy and demanding the information of a lower layer hierarchy sent from said managed system according to the information of the upper layer hierarchy; and
    - transmitting, by said managed system, the information read from its database to said managing system in response to the demand by the managing system and setting up, by said managing system, the database in the lower layer in the previously set up management information tree configuration according to the information transmitted from the managed system.
2. A method of synchronization applied to databases in a network management system according to claim 1, further comprising, at the time when reconfiguring a database due to system reconfiguration of said managed system, the steps of:
    - notifying, by said managed system, the above system reconfiguration to said managing system;
    - demanding, by said managing system, the lower layer managed object instance information required due to the reconfiguration of the database in said managed system;
    - transmitting, by said managed system, the lower layer managed object instance information read from its database to the managing system in response to the demand; and
    - reconfiguring, by said managing system, its database based on the received lower layer managed object instance information.
3. A method of synchronization applied to databases in a network management system according to claim 1, further comprising, at the time when reconfiguring a database due to system reconfiguration of said managed system, the steps of:
    - notifying, by said managed system, the system reconfiguration and backup management information to said managing system; and
    - reconfiguring, by said managing system, its database by using its own backup information referring to said backup management information.

4. A method of synchronization applied to databases in a network management system according to claim 1, further comprising, at the restart of the system of the managed system, the steps of:
    - initially setting, by said managed system, its database in correspondence with the restart level and notifying said managing system of the restart of the system and the restart level; and
    - initially setting, by said managing system, its database in correspondence with the restart level.
5. A method of synchronization applied to databases in a network management system according to claim 1, further comprising, at the restart of the system of said managed system by new software due to an addition of new functions etc. thereto, the steps of:
    - reconfiguring, by said managed system, its database and notifying said managing system of the restart of the system by said new software;
    - demanding, by said managing system, the lower layer managed object instance information in the management information tree configuration, sent from said managed system, corresponding to said addition of the new functions;
    - transmitting, by said managed system, the lower layer managed object instance information read from its database to said managing system in response to said demand; and
    - reconfiguring, by said managing system, its database based on the transmitted lower layer managed object instance information.
6. A method of synchronization applied to databases in a network management system according to claim 1, further comprising, at the restart of the system of said managed system by new software due to an upgraded version etc., the steps of:
    - reconfiguring, by said managed system, its database and notifying said managing system of both the restart of the system by the new software and the update level of the database;
    - demanding, by said managing system, the instance information, sent from said managed system, in correspondence with the update level;
    - transmitting, by said managed system, the instance information read from its database to said managing system in response to that demand; and
    - reconfiguring, by said managing system, its database based on the read instance information or matching up the databases.
7. A method of synchronization applied to databases in a network management system according to claim 1, further comprising, at the restart of the system of said managed system due to a shift over to a new software, the steps of:

- reconfiguring, by said managed system, its database and notifying, to said managing system, of the restart of the system by the new software;
- demanding, by said managing system, differential records, sent from said managed system, in its database;
- seeking, by said managed system, for differential records between the new database and old database and transmitting the sought differential records to said managing system; and

- updating, by said managing system, its database based on said differential records.

Description:

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to a method of synchronization applied to databases in a network management system for establishing synchronization between a database of a managing system and a database of a managed system.

Network management can be classified into for example management of the configuration for managing the constituent elements of a network, management of faults for the detection, analysis, notification, and restoration etc, of faults, management of performance for managing response performance etc., management of security for preventing unauthorized access etc., and management of costs for managing the amount of use of lines etc. A variety of management methods have been already proposed in correspondence with the different sizes of networks etc. For example, a method of managing a network individually for each network element such as exchanges and multiplexers and a method of centrally managing a plurality of network elements are known, and generally such managements are achieved by setting up the databases independently.

In a network management system by treating network elements etc. as managed systems and by providing a managing system for centrally managing these managed systems, synchronizing processing becomes necessary for matching the content of the database set up in the managed system with the content in correspondence with the managed system set in the database of the managing system.

2. Description of the Related Art

Various methods have been already proposed as the method of synchronization for making the contents of a plurality of distributed databases consistent with each other. For example, there is a method in which, when a request is made for updating a database, that database is locked to reject requests for update and after the update of the database, the lock is released. In this method, however, the above mentioned lock is carried out every time the request for updating the database is generated, therefore there was a problem that the processing became complicate. Further, a method has been known in which a time stamp is given to every record in the database, the time stamp is referred to at the time of update of the database, and the update is performed using the record given the newest time stamp.

Further, there also exists a system in which a central computer and a remote computer each having its own database are connected by communication lines and records given time stamps are stored in each database. At the time of update of the database, a record

having a newer time stamp in comparison with the time stamp contained in the request for update information for the database is read, this record is compressed to an update information file and transmitted, and this update information file is stored once in a buffer and then opened up in the database to perform the update processing

In the prior art method of synchronization of databases, a request for updating one database is generated in response to an update of another database in which the update of the former database is carried out in, for example, record units. In order to reduce the frequency of transfers of data in this case, in the related art, the transfer data was compressed to store into the file.

A system having a plurality of network elements such as exchanges and multiplexers is provided with a database for every network element and a managing system for managing these network elements as managed systems. The managing system has its own database storing the same content as the databases for each network element to operate and maintain the system.

Frequently, the network elements as managed systems are made by different manufacturers and are therefore different in configuration. Therefore, the content of the database for each network element as a managed system becomes different. Accordingly, the reconfiguration of the databases in the network elements become, due to the establishment of the database in the managing system and the reconfiguration of the database in the managing system, complex. This becomes a problem in the operation of the system.

6.6.. LET US SUM UP

Therefore, an object of the present invention is to construct databases having the same content in terms of logical configuration irrespective of the configuration of the managed system and thereby facilitate the processing for establishing and reconfiguring database in the managing system.

To attain the above object, the present invention provides a novel method of synchronization applied to databases of a network management system which establishes synchronization between a database of a managing system and a database of a managed system, and thereby facilitates the establishment of an initial database and reconfiguration of the same.

This method comprises (i) establishing a database based on a management information tree modeled (see FIG. 2) on the system configuration for a network element NE forming a managed system; (ii) transferring an upper layer managed object instance information of the management information tree to an operation system OS of the managing system autonomously or by issuance of a command; (iii) starting, by the operation system OS of the managing system, the establishment of the database by the upper layer managed object instance data, (iv) demanding, by this operation system OS, information of a lower layer managed object instance subordinate to the upper layer managed object instance;

(v) sending, by the network element NE of the managed system, the lower layer managed object instance information to the managing system; and (vi) establishing, by the managing system, the database using the thus sent information.

6.7. LESSON END ACTIVITIES

1. Explain about Database design considerations:

6.8. POINTS FOR DISCUSSION

1. Compare Peer-to-Peer and Client-Server Networking

6.9. REFERENCES http://en.wikipedia.org/wiki/Image:Wikibooks-logo-en.svg

- John a . Sharp, ” An introduciton to distributed and parallel processing”, Blackwell Scientific Publicaitons.
- Mukesh Singhal, Shivaratri, “Advanced Concepts in Operating System”, Tata McGraw-Hill, 2001

LESSON 7: CLIENT SERVER NETWORK MODEL

CONTENTS

7.0 Aims and Objectives
7.1. Introduction
7.2. Client-Server Model
7.3. Characteristics
7.4. Architecture
    7.4.1. The Client
    7.4.2. Server Model
7.5.Implementation of Client-Server Model
7.6.Tiered Architecture
7.7.Configuring a Client/Server Network Model
7.8. Let us Sum UP
7.9. Lesson-End Activities
7.10. Points for Discussion
7.11. References

7.0. AIMS AND OBJECTIVES

After studying this lesson, you should be able to

- understand the Client Server Network Model,
- describe the characteristics of the Client Server Model, and
- understand Tired Architecture.

7.1. INTRODUCTION

In the client/server network model a computer plays a centralized role and is known as a server all other computers in the network are known as clients. All client computers access the server simultaneously for files, database, docs, spreadsheets, web pages and resources like hard diver, printer, fax modem, CD/DVD ROM and others. In other words, all the client computes depends on the server and if server fails to respond or crash then networking/communication between the server and the client computes stops.

- The Client – Server Model
- Blocking Vs Non Blocking Primitives
- Buffered Versus Un buffered Primitives
- Implementation of Client – Sever Model.

7.2. CLIENT-SERVER MODEL

Client/server model is a concept for describing communications between computing processes that are classified as service consumers (clients) and service providers (servers). Figure (a) presents a simple C/S model. The basic features of a C/S model are:

1. Clients and servers are functional modules with well defined interfaces (i.e., they hide internal information). The functions performed by a client and a server can be implemented by a set of software modules, hardware components, or a combination thereof.
2. Each client/server relationship is established between two functional modules when one module (client) initiates a service request and the other (server) chooses to respond to the service request. For a given service request, clients and servers do not reverse roles (i.e., a client stays a client and a server stays a server). However, a server for SR R1 may become a client for SR R2 when it issues requests to another server (see Figure). For example, a client may issue an SR that may generate other SRs.
3. Information exchange between clients and servers is strictly through messages (i.e., no information is exchanged through global variables). The service request and additional information is placed into a message that is sent to the server. The server's response is similarly another message that is sent back to the client. This is an extremely crucial feature of C/S model.

    - A computer that has access to services over a computer network. The computer providing the services is a server.

      Client-Server Architecture: An information-passing scheme that works as follows: a client program, such as Mosaic, sends a request to a server. The server takes the request, disconnects from the client and processes the request. When the request is processed, the server reconnects to the client program and the information is transferred to the client. This architecture differs from traditional Internet databases where the client connects to the server and runs the program from the remote site.

4. Messages exchanged are typically interactive. In other words C/S model does not support an off-line process. There are a few exceptions. For example, message queuing systems allow clients to store messages on a queue to be picked up asynchronously by the servers at a later stage.
5. Clients and servers typically reside on separate machines connected through a network. Conceptually, clients and servers may run on the same machine or on separate machines.

The implication of the last two features is that C/S service requests are real-time messages that are exchanged through network services. This feature increases the appeal

of the C/S model (i.e., flexibility, scalability) but introduces several technical issues such as portability, interoperability, security, and performance.

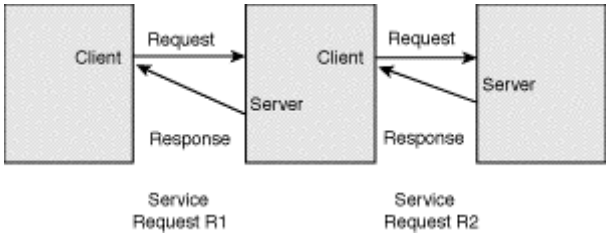


Figure 1: Conceptual Client/Server Model

7.3. CHARACTERISTICS

Characteristics of a Client

- **Request sender is known as client**
- Initiates requests
- Waits for and receives replies.
- Usually connects to a small number of servers at one time
- Typically interacts directly with end-users using a graphical user interface

Characteristics of a Server

- **Receiver of request which is send by client is known as server**
- Passive (slave)
- Waits for requests from clients
- Upon receipt of requests, processes them and then serves replies
- Usually accepts connections from a large number of clients
- Typically does not interact directly with end-users

Comparison to Peer-to-Peer Architecture

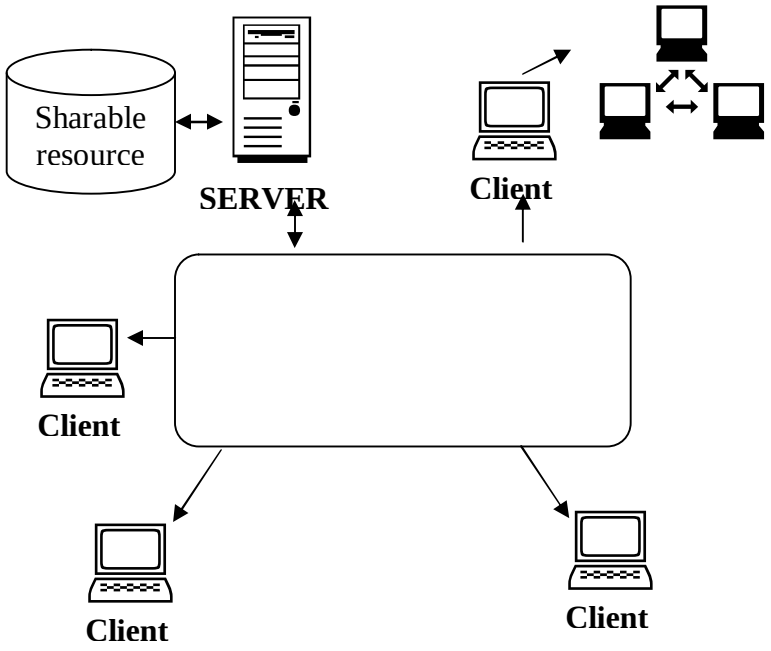
Another type of network architecture is known as peer-to-peer, because each node or instance of the program can simultaneously act as both a client and a server, and because each has equivalent responsibilities and status. Peer-to-peer architectures are often abbreviated using the acronym *P2P*.

Both client-server and P2P architectures are in wide usage today.

7.4. ARCHITECTURE

7.4.1. The Client

In the client-server model, processes are categorized as servers and clients. Servers are the processes that provide services. Processes that need services are referred to as clients. In the client-server model, a client process needing a service (e.g. reading data from a file) sends a message to the server and waits for a reply message. The server process,



Server manages a shareable resource used by a number of clients in the network. The users log on to the client machines.

after performing the requested task, sends the result in the form of a reply message to the client process. Note that servers merely respond to the request of the clients, and do not typically initiate conversations with clients. In systems with multiple serves, it is desirable that when providing services to clients, the locations and conversations among the servers are transparent to the clients. Clients typically make use of a cache to minimize the frequency of sending data requests to the servers. Systems structured on the client-server model can easily adapt to the collective kernel structuring technique.

7.4.2. Server Model

Client-server is software architecture in which to processes interact as superior and subordinate. The client process always initiates requests, and the server process always responds to request. Theoretically, client and server can reside on the same machine. Typically, however, they run on separate machines linked by a local area network

Many different types of server exists, for example:

1. Mail servers.
2. Print servers.

3. File Servers.
4. Database (SQL) Servers

Most people tend to equate the term client-server with a network of PCs or workstations connected by a network to a remote machine running a database server.

7.5. IMPLEMENTATION OF CLIENT – SEVER MODEL

The client/server model is a form of distributed computing where one program (the client) communicates with another program (the server) for the purpose of exchanging information.

The client's responsibility is usually to:

1. Handle the user interface.
2. Translate the user's request into the desired protocol.
3. Send the request to the server.
4. Wait for the server's response.
5. Translate the response into "human-readable" results.
6. Present the results to the user.

The server's functions include:

1. Listen for a client's query.
2. Process that query.
3. Return the results back to the client.

A typical client/server interaction goes like this:

1. The user runs client software to create a query.
2. The client connects to the server.
3. The client sends the query to the server.
4. The server analyzes the query.
5. The server computes the results of the query.
6. The server sends the results to the client.
7. The client presents the results to the user.
8. Repeat as necessary.

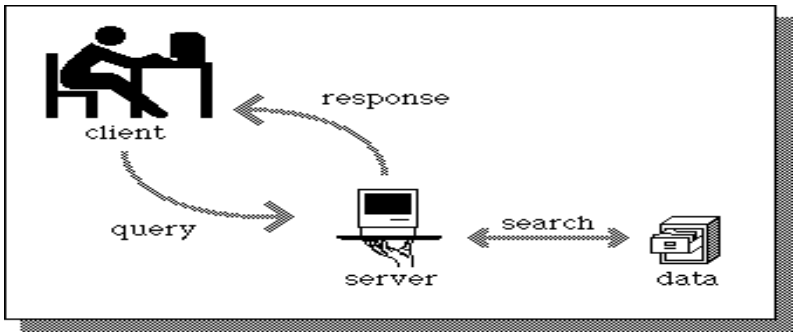


Figure 2: ***A typical client/server interaction***

Flexible user interface development is the most obvious advantage of client/server computing. It is possible to create an interface that is independent of the server hosting the data. Therefore, the user interface of a client/server application can be written on a Macintosh and the server can be written on a mainframe. Clients could be also written for DOS- or UNIX-based computers. This allows information to be stored in a central server and disseminated to different types of remote computers. Since the user interface is the responsibility of the client, the server has more computing resources to spend on analyzing queries and disseminating information. This is another major advantage of client/server computing; it tends to use the strengths of divergent computing platforms to create more powerful applications.

In short, client/server computing provides a mechanism for disparate computers to cooperate on a single computing task.

7.6. TIERED ARCHITECTURE

Generic client/server architecture has two types of nodes on the network: clients and servers. As a result, these generic architectures are sometimes referred to as "two-tier" architectures.

Some networks will consist of three different kinds of nodes: server, application servers which process data for the clients and database servers which store data for the application servers. This is called three-tier architecture.

The advantage of an n-tier architecture compared with a two-tier architecture (or a three-tier with a two-tier) is that it separates out the processing that occurs to better balance the load on the different servers; it is more scalable. The disadvantages of n-tier architectures are:

1. It puts a more load on the network.
2. It is much more difficult to program and test software than in two-tier architecture because more devices have to communicate to complete a users transaction.

First generation systems are 2-tiered architectures where a client presents a graphical user interface (GUI) to the user, and acts according to the user's actions to perform requests of a database server running on a different machine. 2-Tier Architectures.

Client/server applications started with a simple, 2-tiered model consisting of a client and an application server. The most common implementation is a 'fat' client - 'thin' server architecture, placing application logic in the client. (Figure 3) The database simply reports the results of queries implemented via dynamic SQL using a call level interface (CLI) such as Microsoft's Open Database Connectivity (ODBC).

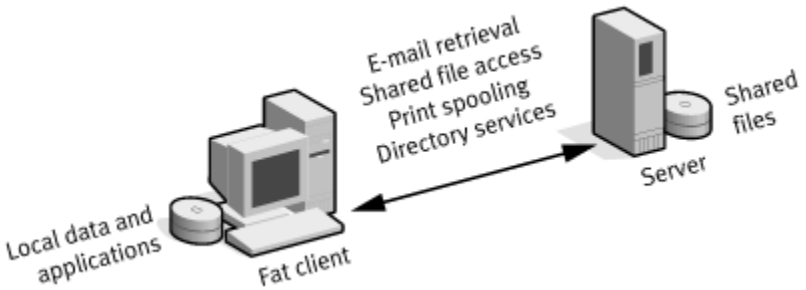


Figure 3: Traditional FAT client/server deployment

Alternate approach is to use thin client - fat server waylays that invokes procedures stored at the database server. (Figure 4) The term thin client generally refers to user devices whose functionality is minimized, either to reduce the cost of ownership per desktop or to provide more user flexibility and mobility.

In either case, presentation is handled exclusively by the client, processing is split between client and server, and data is stored on and accessed through the server. Remote database transport protocols such as SQL-Net are used to carry the transaction.

The network 'footprint' is very large per query so that the effective bandwidth of the network, and thus the corresponding number of users who can effectively use the network, is reduced. Furthermore, network transaction size and query transaction speed is slowed by this heavy interaction. These architectures are not intended for mission critical applications

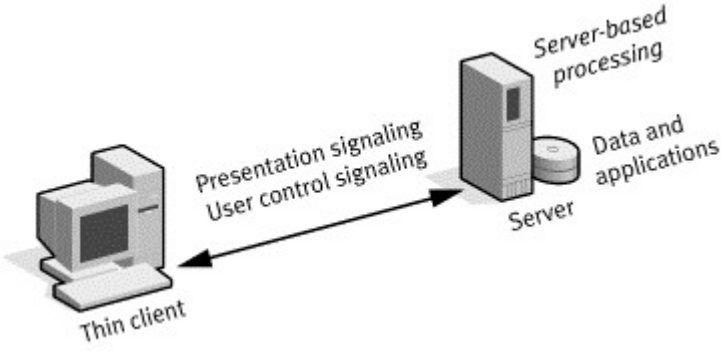


Figure 4: Thin Client/Server Deployment

Development tools that generate 2-tiered fat client implementations include PowerBuilder, Delphi, Visual Basic, and Uniface. The fat server approach, using stored

procedures is more effective in gaining performance, because the network footprint, although still heavy, is lighter than that of a fat client.

Advantages of 2-Tier System

1. Good application development speed
2. Most tools for 2-tier are very robust
3. Two-tier architectures work well in relatively homogeneous environments with fairly static business rules

A new generation of client/server implementation takes this a step further and adds a middle tier to achieve a '3-tier' architecture. Generally, client-server can be implemented in an 'N-tier' architecture where application logic is partitioned. This leads to faster network communications, greater reliability, and greater overall performance.

3-Tier Architectures

Enhancement of network performance is possible in the alternative 'N-tier' client-server architecture. Inserting a middle tier in between a client and server achieves a 3-tier configuration. The components of three-tiered architecture are divided into three layers: a presentation layer, functionality layer, and data layer, which must be logically separate.

(Figure 5) The 3-tier architecture attempts to overcome some of the limitations of 2-tier schemes by separating presentation, processing, and data into separate distinct entities. The middle-tier servers are typically coded in a highly portable, non-proprietary language such as C.

Middle-tier functionality servers may be multithreaded and can be accessed by multiple clients, even those from separate applications.

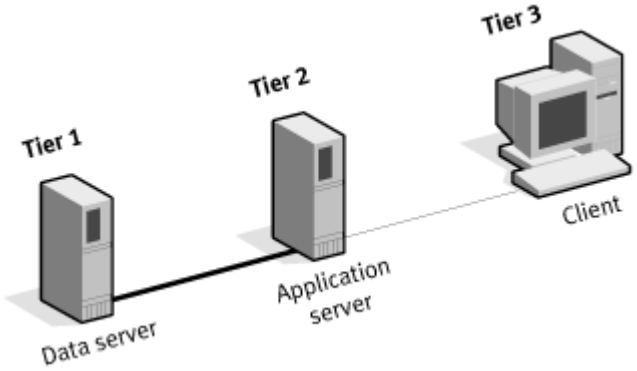


Figure 5. 3-Tiered Application Architecture

The client interacts with the middle tier via a standard protocol such as DLL, API, or RPC. The middle-tier interacts with the server via standard database protocols. The middle-tier contains most of the application logic, translating client calls into database queries and other actions, and translating data from the database into client data in return.

If the middle tier is located on the same host as the database, it can be tightly bound to the database via an embedded 3gl interface. This yields a very highly controlled and high performance interaction, thus avoiding the costly processing and network overhead of SQL-Net, ODBC, or other CLIs.

Furthermore, the middle tier can be distributed to a third host to gain processing power capability.

Advantages of 3-Tier Architecture

1. RPC calls provide greater overall system flexibility than SQL calls in 2-tier architectures
2. 3-tier presentation client is not required to understand SQL. This allows firms to access legacy data, and simplifies the introduction of new data base technologies
3. Provides for more flexible resource allocation Modularly designed middle-tier code modules can be reused by several applications
4. 3-tier systems such as Open Software Foundation's Distributed Computing Environment (OSF/DCE) offers additional features to support distributed applications development

As more users access applications remotely for business-critical functions, the ability of servers to scale becomes the key determinant of end-to-end performance. There are several ways to address this ever-increasing load on servers. Three techniques are widely used:

- ✓ Upsizing the servers
- ✓ Deploying clustered servers
- ✓ Partitioning server functions into a "tiered" arrangement

N-Tier Architectures

The 3-tier architecture can be extended to N-tiers when the middle tier provides connections to various types of services, integrating and coupling them to the client, and to each other. Partitioning the application logic among various hosts can also create an N-tiered system. Encapsulation of distributed functionality in such a manner provides significant advantages such as reusability, and thus reliability.

As applications become Web-oriented, Web server front ends can be used to offload the networking required to service user requests, providing more scalability and introducing points of functional optimization. In this architecture (Figure 6), the client sends HTTP requests for content and presents the responses provided by the application system.

On receiving requests, the Web server either returns the content directly or passes it on to a specific application server. The application server might then run CGI scripts for dynamic content, parse database requests, or assemble formatted responses to client

queries, accessing dates or files as needed from a back-end database server or a file server.

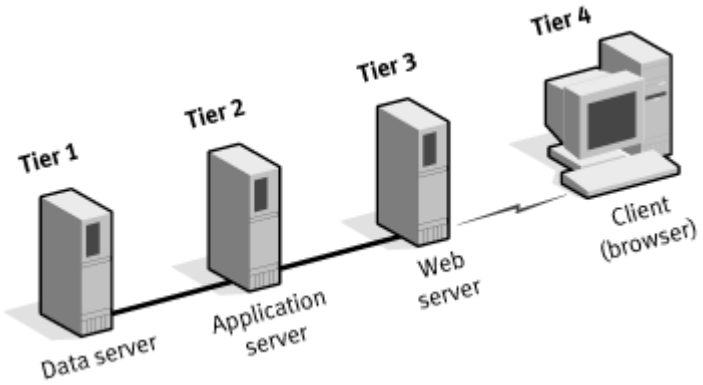


Figure 6. Web-Oriented N-Tiered Architecture

By segregating each function, system bottlenecks can be more easily identified and cleared by scaling the particular layer that is causing the bottleneck.

For example, if the Web server layer is the bottleneck, multiple Web servers can be deployed, with an appropriate server load-balancing solution to ensure effective load balancing across the servers (Figure 7).

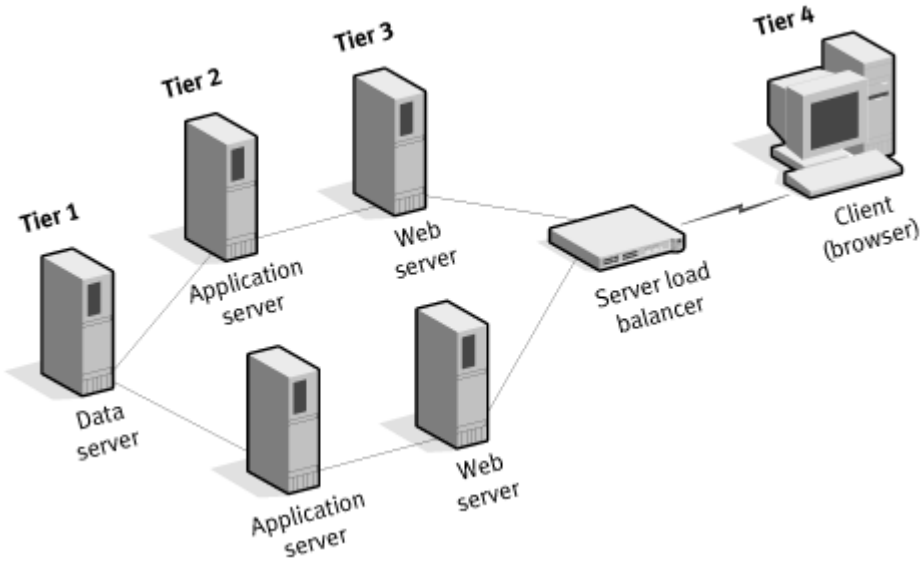
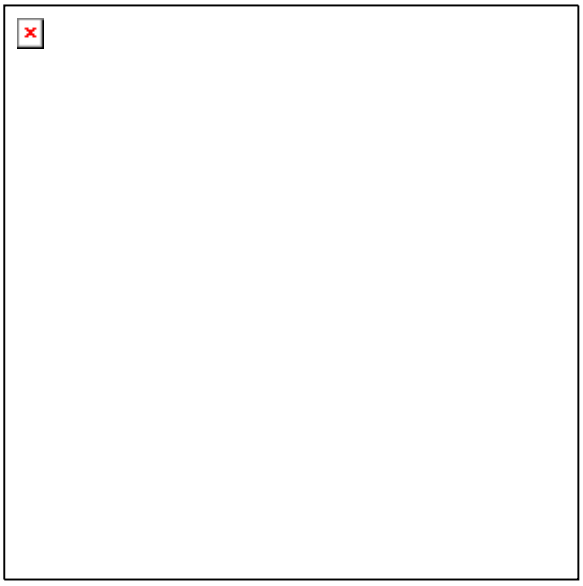
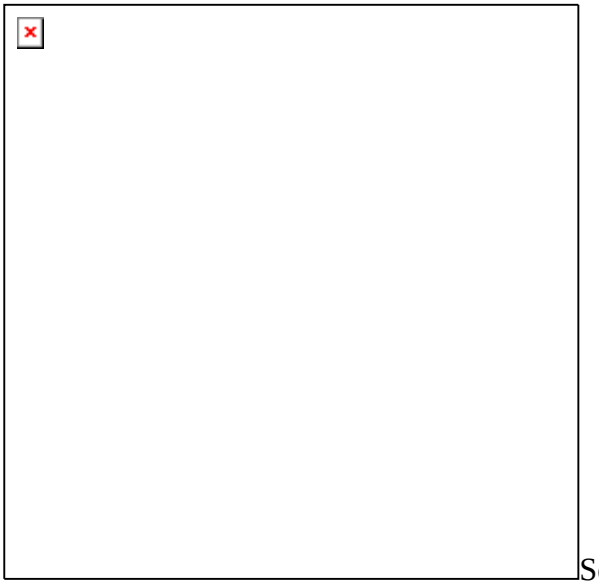
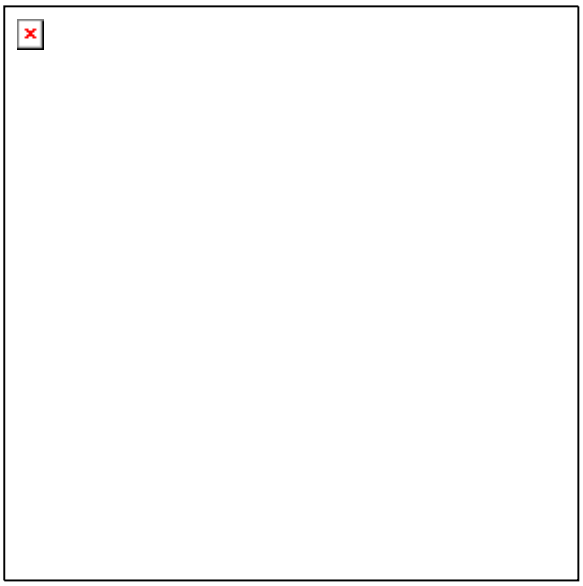


Figure 7. Four-Tiered Architecture with Server Load Balancing

The N-tiered approach has several benefits:

1. Different aspects of the application can be developed and rolled out independently
2.  Servers can be optimized separately for database and application server functions
3.  Servers can be sized appropriately for the requirements of each tier of the architecture

4.  More overall server horsepower can be deployed
5. They are far more scalable, since they balance and distribute the processing load among multiple, often redundant, specialized server nodes.
6. It turn improves overall system performance and reliability, since more of the processing load can be accommodated simultaneously.

The disadvantages of *n*-tiered architectures include:

1. More load on the network itself, due to a greater amount of network traffic.
2. More difficult to program and test than in two-tier architectures because more devices have to communicate in order to complete a client's request.

Comparison to Client-Queue-Client Architecture

While classic Client-Server architecture requires one of communication endpoints to act as a server, which is much harder to implement, Client-Queue-Client allows all endpoints to be simple clients, while the server consists of some external software, which also acts as passive queue (one software instance passes its query to another instance to queue, e.g. database, and then this other instance pulls it from database, makes a response, passes it to database etc.). This architecture allows greatly simplified software implementation. Peer-to-Peer architecture was originally based on Client-Queue-Client concept.

Advantages

- In most cases, a client-server architecture enables the roles and responsibilities of a computing system to be distributed among several independent computers that are known to each other only through a network. This creates an additional advantage to this architecture: greater ease of maintenance. For example, it is

possible to replace, repair, upgrade, or even relocate a server while its clients remain both unaware and unaffected by that change. This independence from change is also referred to as *encapsulation*.
- All the data are stored on the servers, which generally have far greater security controls than most clients. Servers can better control access and resources, to guarantee that only those clients with the appropriate permissions may access and change data.
- Since data storage is centralized, updates to those data are far easier to administer than would be possible under a P2P paradigm. Under a P2P architecture, data updates may need to be distributed and applied to each "peer" in the network, which is both time-consuming and error-prone, as there can be thousands or even millions of peers.
- Many mature client-server technologies are already available which were designed to ensure security, 'friendliness' of the user interface, and ease of use.
- It functions with multiple different clients of different capabilities.

Disadvantages

- Traffic congestion on the network has been an issue since the inception of the client-server paradigm. As the number of simultaneous client requests to a given server increases, the server can become severely overloaded. Contrast that to a P2P network, where its bandwidth actually increases as more nodes are added, since the P2P network's overall bandwidth can be roughly computed as the sum of the bandwidths of every node in that network.
- The client-server paradigm lacks the robustness of a good P2P network. Under client-server, should a critical server fail, clients' requests cannot be fulfilled. In P2P networks, resources are usually distributed among many nodes. Even if one or more nodes depart and abandon a downloading file, for example, the remaining nodes should still have the data needed to complete the download.

Examples

Imagine you are visiting eCommerce web site. In this case, your computer and web browser would be considered the *client*, while the computers, databases, and applications that make up the online store would be considered the *server*. When your web browser requests specific information from the online store, the server finds all of the data in the database needed to satisfy the browser's request, assembles that data into a web page, and transmits that page back to your web browser for you to view.

Specific types of clients include web browsers, email clients, and online chat clients.

Specific types of servers include web servers, application servers, database servers, mail servers, file servers, print servers, and terminal servers. Most web services are also types of servers.

Notes

1. This form of scalability is called *horizontal scalability*. There is substantial and growing criticism that horizontal scalability is limiting as applications become more complex and interdependent, particularly in the areas of network latency, reliability, and manageability. IBM, in particular, takes this view and promotes both vertical and horizontal scalability. Vertical scalability implements fewer servers able to support multiple application and database tiers, and multiple applications, concurrently. The IBM System z is the most notable example of a vertically scalable system.

7.7. CONFIGURING A CLIENT/SERVER NETWORK MODEL

If you want to configure a client-server network model then first prepare the server. Install Windows 2000 or Windows 2003 Server from the CD on the server computer and make a domain. You can create a domain by this command on the Run "DCPROMO". You can give this command once you install the server successfully. After you give the DCPROMO command you will be asked for a unique domain name. All the client computers will use the same unique domain name for becoming the part of this domain. This command will install the active directory on the server, DNS and other required things. A step by step wizard will run and will guide you for the rest of the steps. Make sure that a network cable is plugged in the LAN card of the server when you run the DCPROMO.exe command.

When the Active directory is properly installed on the server, restart the server. You can create network users on the server computer and also name/label the network resources like computers/printers etc.

Once you install the server successfully now come to the client computers. Install Windows 2000 professional on your all client computers. Once you install the Windows 2000 professional on the clients the next step is to make this computer (client computer) a part of the network.

Configuration Steps

1. Choose a unique name for each client computer
2. Choose unique IP address for each computer and relevant.
3. Use the same domain name for all client PCs.

Network/System administrators are required to do these administrative tasks on the server and client computers. Any shared resources on the network either on the server or the clients can be access through the My Network Places in the Windows 2000 platform. There is another way to connect to the shared resources by giving this command in the run \\ComputerName\SharedDriveLetter.

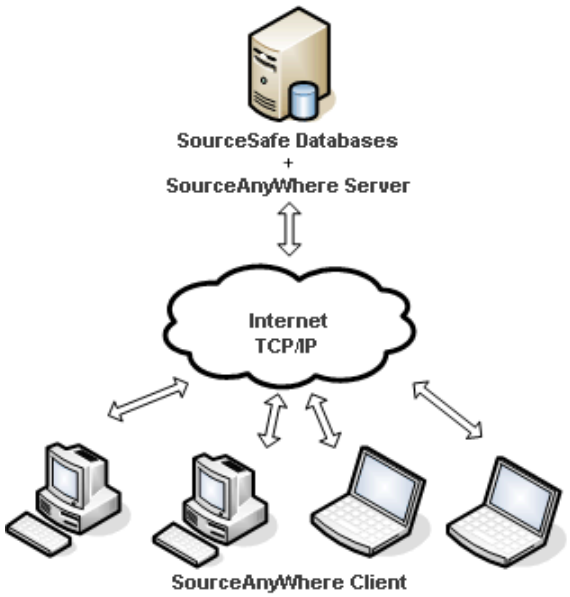
Network configurations steps can be implemented by right clicking the My Computer>Properties>

For giving the IP address you will have to right click on the My Network places>properties>Local Area Connection>Properties>Internet Protocols (TCP/IP)>Properties and then give the IP address and subnet mask of the same range and class for all the computers in the network.

7.8. LET US SUM UP

Client/Server is a type of network in which particular computers "servers" are responsible for resources and providing these resources for other computers "clients". There can be many clients and one server, or vice versa. The client makes request to the server for some service and in return the server gives response to the client.

Client/Server is a scalable architecture whereby each computer or process on the network is either a client or a server. Server software generally, but not always, runs on powerful computers dedicated for exclusive use for running the business application. Client software on the other hand generally runs on common PCs or workstations. Clients get all or most of their information and rely on the application server for things such as configuration files, stock quotes, business application programs, data from databases etc.



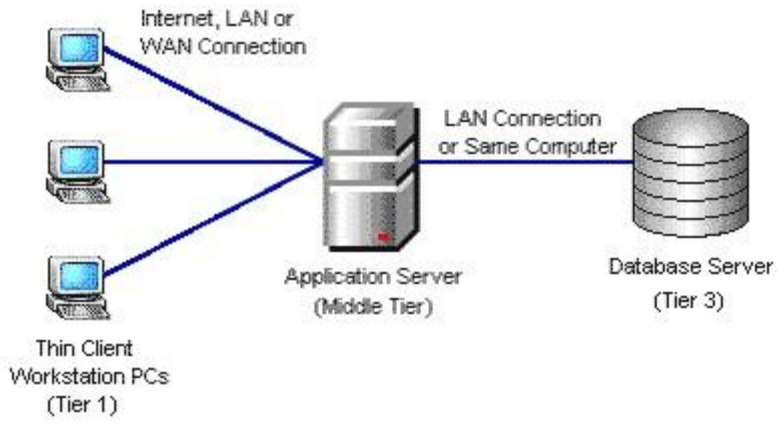
Multi tired is an architecture in which in which we divide the network based on their functionalities. In multi tier architecture we have several tiers based on their specific functionalities. In this architecture middle ware is called middle tier. It schedules the request send from client among different computers of the network. Each system is a separate tier according to the functionality it provides. For example the middle tier may send the request to application Server that is a separate tier, application server may send

the request to data Server for processing that will be another tier. The communication from tier to tier is continued until clients request is processed.

Multi-tier Architecture gives a single system image to client by hiding internal complexities of job assignment from tier to tier. Thus transparency of system is achieved.

A 3-tier application is an application program that is organized into three major parts, each of which is distributed to a different place or places in a network. The three parts are:

- The workstation or presentation interface
- The business logic
- The database and programming related to managing it



7.9. LESSON END ACTIVITIES

1. Explain the client/server model.
2. Explain the Blocking verses un-buffered primitives
3. Explain the implementation of client/server model.
4. Explain the characteristics of communications in distributed systems.

7.10. POINTS FOR DISCUSSION

1. Discuss about various Applications of Client Server Architecture

7.11. REFERENCES http://en.wikipedia.org/wiki/Image:Wikibooks-logo-en.svg

- John a . Sharp, ” An introduciton to distributed and parallel processing”, Blackwell Scientific Publicaitons.
- Mukesh Singhal, Shivaratri, “Advanced Concepts in Operating System”, Tata McGraw-Hill, 2001

LESSON 8: TYPES OF SERVERS

CONTENTS

8.0 Aims and Objectives
8.1. Introduction
8.2. File Server
    8.2.1. File and print
    8.2.2. Security
8.3. Print Server
8.4. Mail Server
8.4.1. SMTP
8.4.2. POP/IMAP
8.4.3. Mail filtering
8.4.4. Comparison of mail servers
8.5. Let us Sum UP
8.6. Lesson-End Activities
8.7. Points for Discussion
8.8. References

8.0. AIMS AND OBJECTIVES

After studying this lesson, you should be able to:

- Describe the various types of Servers
- Understand the uses of File, Print and Mail Server.

8.1. INTRODUCTION

Client Server Architecture comprises of various types of servers, such as File Server, Print Server, and Email Servers. In this lesson, we will be discussing about these servers and various protocols used in Mail Servers, we also see a comparative study of various mail servers.

8.2. FILE SERVER

In telecommunication, the term **file server** has the following meanings:

- In the client/server model, a file server is a computer responsible for the central storage and management of data files so that other computers on the same network can access the files. A file server allows users to share information over a network without having to physically transfer files by floppy diskette or some other external storage device. Any computer can be configured to be a host and act as a file server. In its simplest form, a file server may be an ordinary PC that handles requests for files and sends them over the network. In a more sophisticated network, a file server might be a dedicated network-attached storage

(NAS) device that also serves as a remote hard disk drive for other computers, allowing anyone on the network to store files on it as if to their own hard drive.

- A form of disk storage that hosts files within a network; file servers do not need to be high-end but must have enough disk space to incorporate a large amount of data. Many people mistake file servers for a high-end storage system, but in reality, file servers do not need to possess great power or super fast computer specifications.

- A computer program that allows different programs, running on other computers, to access the files of that computer

- In common parlance, the term **file server** refers specifically to a computer on which a user can map or mount a disk drive or directory so that the directory appears to be on the machine at which the user is sitting. Additionally, on this type of file server, the user can read or write a file as though it were part of the file system of the user's computer.

Files and directories on the remote computer are usually accessed using a particular protocol, such as WebDAV, SMB, CIFS, NFS, Appletalk or their mutations.

- Although files can be sent to and received from most other computers unless their primary function is access by the above means, they are generally not considered file servers as such.

8.2.1. File and print

Traditionally, file and print services have been combined on the same computers due to similar computing requirements for both functions. Usually, such computers are distinct from application and database servers, which have different, usually more processor-intensive, requirements. However, as computing power increases and file serving requirements remain relatively constant, it is more common to see these functions combined on the same machine.

8.2.2. Security

File servers generally offer some form of system security to limit access to files to specific users or groups. In large organizations, this is a task usually delegated to what is known as directory services such as Novell's eDirectory or Microsoft's Active Directory.

These servers work within the hierarchical computing environment which treat users, directories, computers, applications and files as distinct but related entities on the network and grant access based on user or group credentials. In many cases, the directory service spans many file servers, potentially hundreds for large organizations. In the past, and in smaller organizations, authentication can take place directly to the server itself.

8.3. PRINT SERVER

A **print server**, or printer server, is a computer or device to which one or more printers are connected, which can accept print jobs from external client computers connected to the print server over a network. The print server then sends the data to the appropriate printer that it manages. The term **print server** can refer to:

1. A host computer running Windows OS with one or more shared printers. Client computers connect using Microsoft Network Printing protocol.
2. A computer running some operating system other than Windows, but still implementing the Microsoft Network Printing protocol (typically Samba running on a UNIX or Linux computer).
3. A computer that implements the LPD service and thus can process print requests from LPD clients.
4. A dedicated device that connects one or more printers to a LAN. It typically has a single LAN connector, such as an RJ-45 socket, and one or more physical ports (e.g. serial, parallel or USB (Universal Serial Bus)) to provide connections to printers. In essence this dedicated device provides printing protocol conversion from what was sent by client computers to what will be accepted by the printer. Dedicated print server devices may support a variety of printing protocols including LPD/LPR over TCP/IP, NetWare, NetBIOS/NetBEUI over NBF, TCP Port 9100 or RAW printer protocol over TCP/IP, DLC or IPX/SPX. Dedicated server appliances do not provide spooling or print queue services, since they typically have very little memory.
5. A dedicated device similar to definition 4 above, that also implements Microsoft Networking protocols to appear to Windows client computers as if it were a print server defined in 1 above.

The term **print server** normally has the meaning defined in 1 or 2 above, while the term **print server device** usually refers to definition 4.

8.4. MAIL SERVER

A **mail transfer agent** or **MTA** (also called a **mail transport agent**, **message transfer agent**, **mail server**, **SMTPD** (short for SMTP daemon), or a **mail exchanger** (MX) in the context of the Domain Name System) is a computer program or software agent that transfers electronic mail messages from one computer to another.

It receives messages from another MTA (relaying), a mail submission agent (MSA) that itself got the mail from a mail user agent (MUA), or directly from an MUA, thus acting as an MSA itself. The MTA works behind the scenes, while the user usually interacts with the MUA.

The delivery of e-mail to a user's mailbox typically takes place via a mail delivery agent (MDA); many MTAs have basic MDA functionality built in, but a dedicated MDA like procmail can provide more sophistication.

According to various surveys the most popular mail server software are sendmail, Postfix, Microsoft Exchange Server, Exim, IMail (by Ipswitch, Inc.), MDaemon by Alt-N Technologies, MailEnable, Merak Mail Server and qmail. The MailChannels survey also found that many organizations use the services of e-mail security services such as Postini, MXLogic or Concentric Hosting to receive e-mail.

This is a **list of mail servers**: mail transfer agents, mail delivery agents, and other computer software which provide e-mail services.

SMTP, POP/IMAP, Mail filtering

8.4.1. SMTP

Apache James, Atmail, AXIGEN, Citadel, CommuniGate Pro, Courier, Eudora Internet Mail Server, Exim, Hexamail server, IBM Lotus Domino, IpBrick, Ipswitch IMail Server ,Kerio MailServer, MailEnable Mail Server, Mailtraq, Merak Mail Server, Mercury Mail Transport System, MeTA1 (successor of the sendmail X project), Microsoft Exchange Server, MMDF, Novell GroupWise, Novell NetMail, Open-Xchange, PostCast Server, Postfix, PostPath Email and Collaboration Server, qmail, Scalix, Sendmail, SmarterMail, SparkEngine, Sun Java System, WinGate, WorkgroupMail, Xmail, XMS Email Application Server, Zimbra, ZMailer

8.4.2. POP/IMAP

Apache James, Axigen, Binc IMAP - uses Maildir, Bluebottle, Citadel - uses a database-driven mail store, CommuniGate Pro, Courier Mail Server - uses Maildir format, Cyrus IMAP server, Dovecot, Eudora Internet Mail Server, Hexamail server, IpBrick, Ipswitch IMail Server, Meldware Communication Server (Free open source [multi-platform] mail server), Kerio MailServer, Lotus Domino IMAP4 Server, MailEnable Mail Server, Mailtraq, Merak Mail Server, Mercury Mail Transport System, Microsoft Exchange Server, Microsoft Windows POP3 Service, Novell GroupWise, Novell NetMail, Open-Xchange, Oryx Archiveopteryx, PostPath, Qpopper, SmarterMail, UW IMAP - uses mbox format, WinGate, WorkgroupMail, Zimbra

8.4.3. Mail filtering

ASSP, Bayesian filters, Bogofilter, DSPAM, Hexamail Guard, maildrop, Mailtraq, Procmail, PureMessage, SurfControl, SpamAssassin, WinGate, WorkgroupMail, Gattaca Serve, Vipul's Razor

8.4.4. Comparison of mail servers

This is a **comparison of mail servers**: mail transfer agents, mail delivery agents, and other computer software which provide e-mail services.

Feature comparison

Mail server	Server OS support			Features								Storage			License
	Linux/Unix	Windows	Mac OS X	SMTP	POP3	IMAP	SMTP over TLS	POP over TLS	NNTP	SSL	Webmail	Database	Filesystem	Other	
Apache James	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	No	Open Source/ASLv2
BincIMAP	Yes	No	Yes	No	No	Yes	No	No	?	Yes	No	No	Yes	No	Open source/GPL
CommuniGate Pro	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	No	Yes	No	Proprietary
Courier Mail Server	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	?	Yes	Yes	Yes	Yes	No	Open Source
Eudora Internet Mail Server	No	No	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No	Yes	Proprietary
Exim	Yes	via	Yes	Yes	No	No	Yes	No	?	Yes	No	No	Yes	No	Open Source/GPL

		Cygwin													
GroupWise	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	?	Yes	Yes	Yes	No	No	Proprietary
Hexamail Server	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	No	No	Yes	Proprietary
IBM Lotus Domino	Yes	Yes	?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Proprietary
Kerio MailServer	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	?	Yes	Yes	No	Yes	No	Proprietary
Mailtraq	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Proprietary
Meldware Mail Server	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	?	Yes	Yes	Yes	Yes	No	Open source/LGPL
Merak Mail Server	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Proprietary
Microsoft Exchange	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	ESE only	No	No	Proprietary

NetMail	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	?	Yes	Yes	Yes	No	No	Proprietary
Postfix	Yes	No	Yes	Yes	No	No	Yes	No	No	Yes	No	Yes	Yes	Yes	Open source/IBM Public License
PostPath	Yes	No	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	**?**	Yes	**?**	Proprietary
Qmail	Yes	No	Yes	Yes	Yes	No	No	No	?	No	No	No	Yes	No	free-to-use
QmailToaster	Yes	No	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	free-to-use
Sendmail	Yes	No	No	Yes	No	No	Yes	No	No	No	No	**?**	Yes	**?**	Open source/Sendmail License
Sun Java System Messaging Server	Yes	No	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	No	Proprietary
UWIMAP	Yes	Yes	Yes	No	Yes	Yes	No	Yes	No	Yes	WebPine	No	Yes	No	Open Source/Apache license

WinGate	No	Yes	No	Yes	Yes	Yes	Yes	Yes	?	Yes	Yes	No	Yes	No	Proprietary
Workgroup Mail	No	Yes	No	Yes	Yes	Yes	No	No	?	Yes	Yes	Yes	No	No	Proprietary
Zimbra	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	?	Yes	Yes	Yes	Yes	No	Open Source/Proprietary

Authentication

Mail server	Login			Authentication			
	SMTP AUTH	POP before SMTP	APOP	Filesystem	Database	LDAP	Other
Apache James	Yes	?	?	?	Yes	Yes	?
CommuniGate Pro	Yes	Yes	Yes	Yes	No	Yes	CLRTXT, CRAM-MD5, DIGEST-MD5, APOP, GSSAPI, NTLM, IMSN, WEBUSER
Courier Mail Server	Yes	Yes	Yes	Yes	Yes	Yes	Pluggable Authentication Modules, userdb, CRAM, LDAP, MySQL, vchkpw/vpopmail, PostgreSQL (beta)
Eudora Internet Mail Server	Yes	No	Yes	Yes	No	No	CRAM-MD5, DIGEST-MD5, PLAIN, LOGIN
Exim	Yes	?	Yes	Yes	Yes	Yes	Cyrus SASL, CRAM-MD5, PLAIN, LOGIN, SPA

GroupWise	Yes	Yes	Yes	Yes	Yes	Yes	eDirectory, Any LDAPv3-compliant source
Hexamail Server	Yes	Yes	Yes	Yes	No	Yes	Active Directory, Any LDAPv3-compliant source, CRAM-MD5, DIGEST-MD5, PLAIN, LOGIN
Kerio MailServer	Yes	Yes	Yes	Yes	No	Yes	Active Directory, Apple Open Directory, ActiveSync, Pluggable Authentication Modules
Mailtraq	Yes	Yes	Yes	Yes	Yes	Yes	NT Domain, Active Directory, Local Directory
Meldware Mail Server	Yes	No	Yes	Yes	Yes	Yes	?
Merak Mail Server	Yes	Yes	Yes	Yes	Yes	Yes	PLAIN, CRAM-MD5, DIGEST-MD5, NT Domain, Active Directory, any LDAP-compatible source
Microsoft Exchange	Yes	?	Yes	No	No	Yes	Active Directory
Net Mail	Yes	Yes	Yes	Yes	Yes	Yes	eDirectory, LDAP

Postfix	Yes	No	No	Yes	Yes	Yes	?
PostPath	Yes	?	?	No	No	Yes	Active Directory
WinGate	Yes	Yes	Yes	Yes	Yes	No	NT Domain, Active Directory, CRAM-MD5, SASL PLAIN, SASL LOGIN
Workgroup Mail	Yes	No	No	No	Yes	Yes	NT Domain, Active Directory
Zimbra	Yes	?	Yes	Yes	Yes	Yes	Internal, LDAP

8.5. LET US SUM UP

In this lesson we saw various types of servers used in Client Server Computing, we also made a descriptive study of various mail servers used.

8.6. LESSON END ACTIVITIES

1. List down various mail server protocol and its role

8.7. POINTS FOR DISCUSSION

1. Discuss the various steps to configure Print Server in Windows Environment

8.8. REFERENCES

http://en.wikipedia.org/wiki/Image:Wikibooks-logo-en.svg

1. Joel M. Crichllow “Introduction to distributed and Parallel Commuting”.

UNIT – V

LESSON 9: DISTRIBUTED DATA BASES

CONTENTS

9.0 Aims and Objectives
9.1 Introduction
9.2. Need for Distributed Database
9.3. Types of Distributed Databases
9.4. Principles of Distributed Databases
9.5. Advantages of Distributed Databases
9.6. Limitations of Distributed Database
9.7.Strategies for Distributing Database
    9.7.1. Fragmentation
    9.7.2. Replication
9.8. Let us Sum UP
9.9. Lesson-End Activities
9.10. Points for Discussion
9.11. References

9.0. AIMS AND OBJECTIVES

After studying this lesson, you should be able to:

- Describe the salient characteristics of the variety of distributed database environments.
- Explain the potential advantages and risk associated with distributed databases.
- State the relative advantages of synchronous and asynchronous data replication and partitioning as three major approaches for distributed database design.

9.1. INTRODUCTION

A distributed database system consists of multiple independent databases that operate on two or more computers that are connected and share data over a network. The databases are usually in different physical locations. Each database is controlled by an independent DBMS, which is responsible for maintaining the integrity of its own databases. In extreme situations that databases might be installed on different hardware, different operating systems, and could even use DBMS software from different vendors. That last contingency is the hardest to handle. Most current distributed databases function better if all of the environments are running DBMS software from the same vendor.

9.2. NEED FOR DISTRIBUTED DATABASE

When an organization is geographically dispersed, it may choose to store its database on a central computer or to distribute them to local computers (or a combinatory of both). The following conditions encourage the need of distributed database in a business organization:

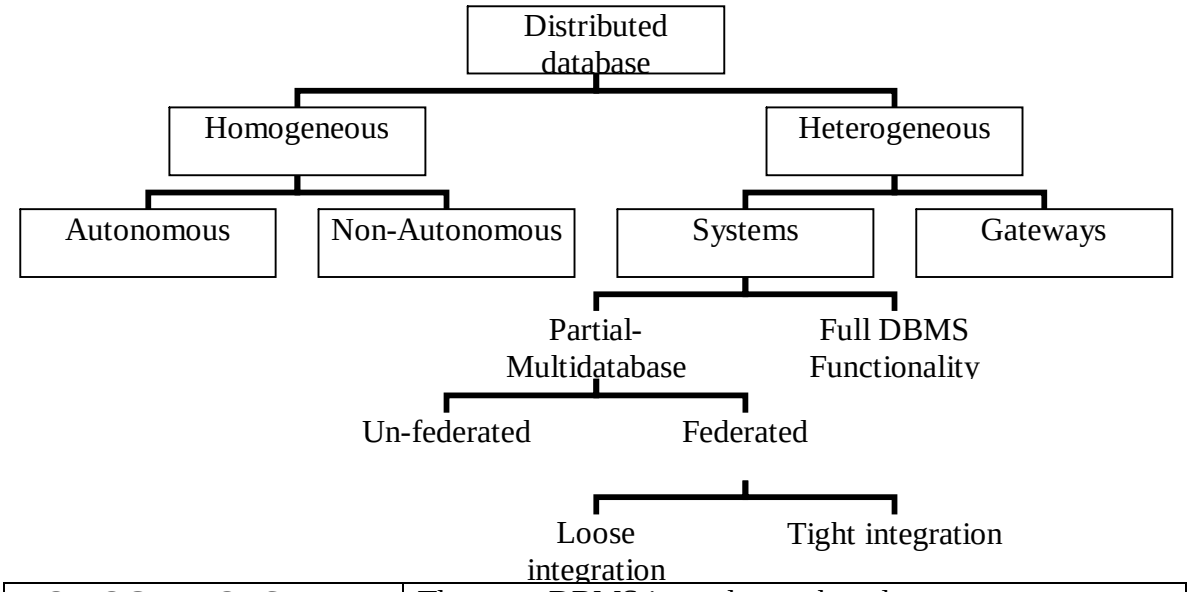
Distribution and autonomy of business units: Divisions, departments, and facilitates in modern organizations are often geographically (and possibly internationally) distributed. Often each unit has the authority to create its own information systems, and often these units want local data over which they can have controls.

Data sharing: Even moderately complex business decisions require sharing data across business units, so it must be convenient to consolidate data across local databases on demand.

Data communications costs and reliability: The cost to ship large quantities of data cross a communications network or to handle a large volume of transactions from remote sources can be high. It is often more economical to locate data and applications close to where they are needed. Also, dependence on data communications can be risky, so keeping local copies or fragments of data can be reliable way to support the need for rapid access to data across the organization.

9.3. TYPES OF DISTRIBUTED DATABASES

The following chart explains briefly about different types of Distributed databases



HOMOGENEOUS	The same DBMS is used at each node
Autonomous	Each DBMS works independently, passing messages back and forth to share data updates.
Non-autonomous	A central, or master, DBMS coordinates database access and update across the nodes.
HETEROGENEOUS	Potentially different DBMSs are used at each node.
Systems	Supports some or all the functionality of one logical database.
Full DBMS Functionality	Supports all of the functionality of a distributed database.
Partial-Multidatabase	Supports some features of a distributed database.
Federated	Supports local databases for unique data requests.
Loose Integration	Many Schemas exists, for each local database, and each local DBMS must communicate with all local schemas.
Tight Integration	One global schema exists that defines all the data cross all local databases.
Unfederated	Requires all access to go through a central coordinating module.
Gateways	Simple paths are created to other databases, without the benefits of one logical database.

9.4. PRINCIPLES OF DISTRIBUTED DATABASES

A distributed database is one that is held in several locations. There are three methods of achieving this:

1. Each remote processor has data on its own customers, stock, etc. The database is uploaded each night and changes made.

2. The entire database is duplicated at the remote site. The database is uploaded each night and changes made.

3. The central database contains only an index of entries. The actual records are held at the remote site. A query to the central database will locate where the record is held. This system is used by very large databases.

Distributed processing is growing rapidly because speed of processing at local sites is increased. The drawbacks are security concerns and the need to maintain data integrity.

A *distributed database* appears to a user as a single database but is, in fact, a set of databases stored on multiple computers. The data on several computers can be simultaneously accessed and modified using a network. Each database server in the distributed database is controlled by its local DBMS, and each cooperates to maintain the consistency of the global database. ***Figure a*** illustrates a representative distributed database system.

The following sections outline some of the general terminology and concepts used to discuss distributed database systems.

Clients, Servers and Nodes

A database *server* is the software managing a database, and a *client* is an application that requests information from a server. Each computer in a system is a *node*. A node in a distributed database system can be a client, a server, or both. For example, in ***Figure a***, the computer that manages the HQ database is acting as a database server when a statement is issued against its own data (for example, the second statement in each transaction issues a query against the local DEPT table), and is acting as a client when it issues a statement against remote data (for example, the first statement in each transaction is issued against the remote table EMP in the SALES database).

Oracle supports heterogeneous client/server environments where clients and servers use different character sets. The character set used by a client is defined by the value of the NLS_LANG parameter for the client session. The character set used by a server is its database character set. Data conversion is done automatically between these character sets if they are different.

Direct and Indirect Connections

A client can connect directly or indirectly to a database server. In **Figure a**, when the client application issues the first and third statements for each transaction, the client is connected directly to the intermediate HQ database and indirectly to the SALES database that contains the remote data.

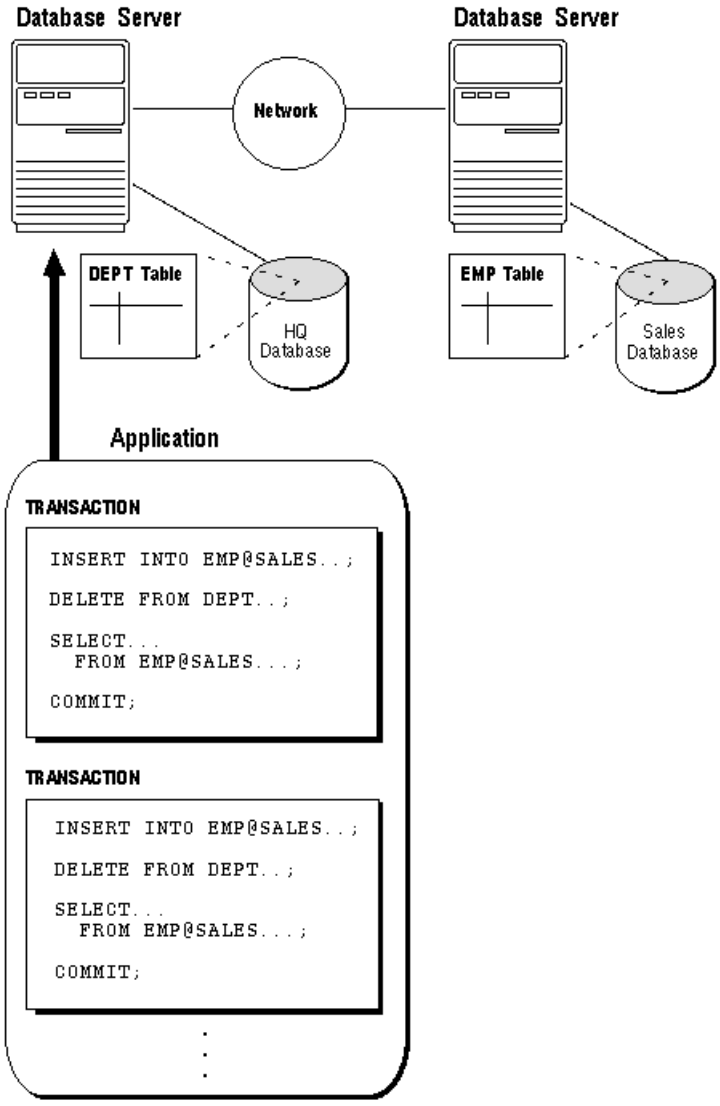


Figure a. An Example of a Distributed DBMS Architecture

Site Autonomy

Site autonomy means that each server participating in a distributed database is administered independently (for security and backup operations) from the other databases, as though each database was a non-distributed database. Although all the databases can work together, they are distinct, separate repositories of data and are administered individually. Some of the benefits of site autonomy are as follows:

- Nodes of the system can mirror the logical organization of companies or cooperating organizations that need to maintain an "arms length" relationship.
- The local database administrator controls local data. Therefore, each database administrator's domain of responsibility is smaller and more manageable.

- Independent failures are less likely to disrupt other nodes of the distributed database. The global database is partially available as long as one database and the network are available; no single database failure need halt all global operations or be a performance bottleneck.
- Failure recovery is usually performed on an individual node basis.
- A data dictionary exists for each local database.
- Nodes can upgrade software independently.

Schema Objects and Naming in a Distributed Database

A schema object (for example, a table) is accessible from all nodes that form a distributed database. Therefore, just as a non-distributed local DBMS architecture must provide an unambiguous naming scheme to distinctly reference objects within the local database, a distributed DBMS must use a naming scheme that ensures that objects throughout the distributed database can be uniquely identified and referenced.

To resolve references to objects (a process called *name resolution*) within a single database, the DBMS usually forms object names using a hierarchical approach. For example, within a single database, a DBMS guarantees that each schema has a unique name, and that within a schema, each object has a unique name. Because uniqueness is enforced at each level of the hierarchical structure, an object's local name is guaranteed to be unique within the database and references to the object's local name can be easily resolved.

Distributed database management systems simply extend the hierarchical naming model by enforcing unique database names within a network. As a result, an object's *global object name* is guaranteed to be unique within the distributed database, and references to the object's global object name can be resolved among the nodes of the system.

For example, ***Figure b*** illustrates a representative hierarchical arrangement of databases throughout a network and how a global database name is formed.

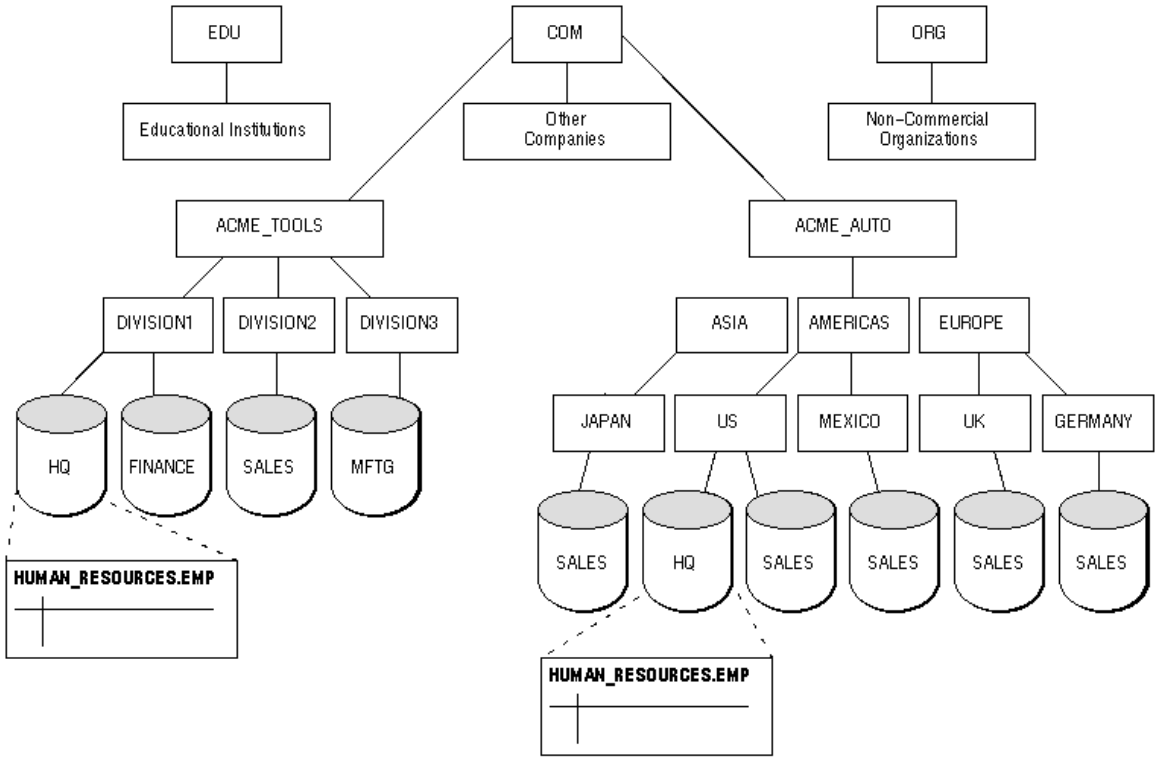


***Figure b*. Network Directories and Global Database Names**

Database Links

To facilitate connections between the individual databases of a distributed database, Oracle uses *database links*. A database link defines a "path" to a remote database.

Database links are essentially transparent to users of a distributed database, because the name of a database link is the same as the global name of the database to which the link points

9.5. ADVANTAGES OF DISTRIBUTED DATABASES

Compared to centralized databases, there are numerous advantages, they are:

Increased reliability and availability: When a centralized system fails, the database is unavailable to all users. A distributed system will continue to function at some reduced level, however, even when a components fails. The reliability and availability will depend (among other things) on how the data are distributed.

Local control: Distributing the data encourages local groups to exercise greater control over “their” data, which promotes improved data integrity and administration. At the same time, user can access non-local data when necessary. Hardware can be chosen for the local site to match the local, not global, data processing work.

Modular growth: Suppose that an organization expands to a new location or adds a new work group. It is often easier and more economical to add a local computer and its associated data to the distributed network than to expand a large central computer. Also, there is less chance of disruption to existing users than is the case when a central computer system is modified or expanded.

Lower communication costs: With a distributed system, data can be located closer to their point of use. This can reduce communication costs, compared to a central system.

Faster response: Depending on how data are distributed, most requests for data by users at a particular site can be satisfied by data stored at that site. This speeds up query processing since communication and central computer delays are minimized. It may also be possible to split complex queries into sub-queries than can be processed in parallel at several sites, providing even faster response.

Other advantages of distributed databases are:

1. Provide faster response times and service at each location.
2. Reduce the vulnerability of putting all essential data at one site
3. Faster response to local queries
4. Reduction in amount of network traffic
5. Effect of breakdowns is minimized
6. Better local control over the system
7. Less powerful cheaper processors needed

9.6. LIMITATIONS OF DISTRIBUTED DATABASE

A distributed database system also faces certain cost and disadvantages:

Software cost and complexity: More complex software (especially the DBMS) is required for a distributed data base environment.

Processing overhead: The various sites must exchange messages and perform additional calculations to ensure proper coordination among data at the different sites.

Data Integrity: A by-product of the increased complexity and need for coordination is the additional exposure to improper updating and other problems of data integrity.

Slow response: If the data are not distributed properly according to their usage, or if queries are not formulated correctly, response to request for data can be extremely slow.

Other Limitations of distributed databases

1. Security - dependence on telecommunications links and widened access to sensitive data.

2. Increase data redundancy especially if a replicated database is chosen. Inconsistencies can easily arise between central and local systems, especially if changes to the data not immediately updated by the other.
3. Increased tendency to data redundancy and data inconsistency.
4. System is dependent on high quality telecommunication lines, which may be vulnerable.
5. Need to maintain and enforce consistent standards and data definitions over a wide area.
6. Increased security problems - need to enforce security procedures over wider area plus increased problems over data transmission.

9.7. STRATEGIES FOR DISTRIBUTING DATABASE

Distribution of data in DDBMS is done by two ways:

a. Fragmentation, b. Replication

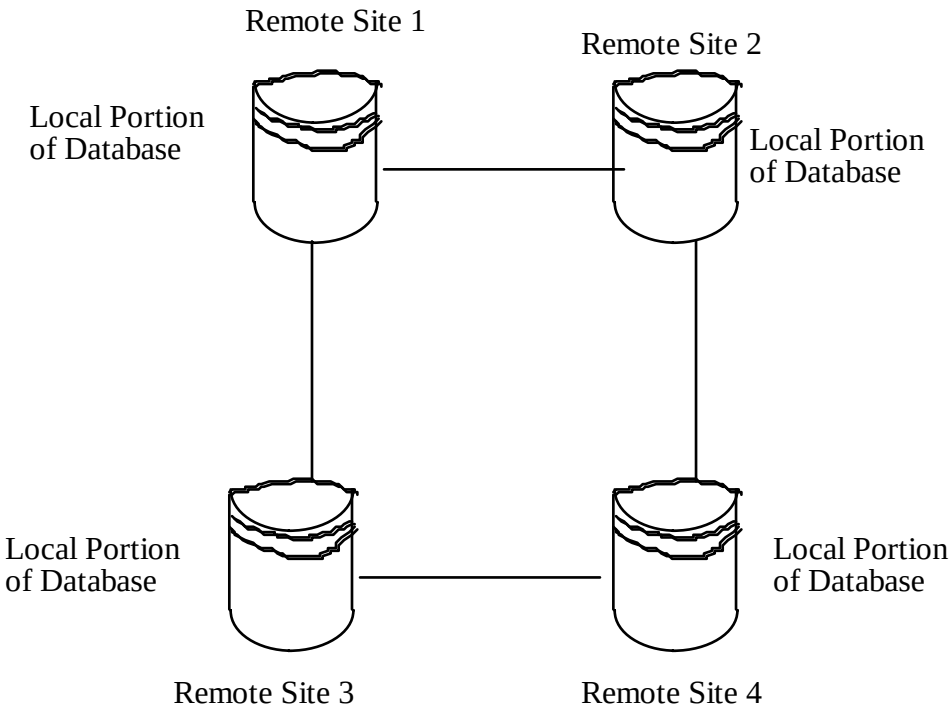
In a distributed DBMS, relations are stored across several sites. Accessing a relation that is stored at a remote site incurs message-passing costs, and to reduce this overhead, a single relation may be partitioned or fragmented across several sites, with fragments stored at the sites where they are most often accessed, or replicated at each site where the relation is in high demand.

9.7.1. Fragmentation

A **Fragmentation** (**partitioned) database** is subdivided so that each location has only the portion that serves its local needs. Partition. The database is partitioned with each node on the network containing that section of the database that relates to it. For example the section of the database that relates to customers served at that node. Other data is held centrally and any changes to central data can be dealt with overnight by a batch update.

An extreme case of this approach is when the entire database is replicated to local processors. Here again the central copy will be updated by batch processing, probably overnight.

Both of these approaches lead to data inconsistency and will therefore be unsuitable for applications such as holiday bookings where data changes at one node need to be available to other nodes. They will however work well in situations where local data processing is compartmentalized and has no immediate effect on other nodes. An example of this is supermarket stock control.



PARTITIONED DATABASE

Fragmentation consists of breaking a relation into smaller relations or fragments, and storing the fragments (instead of the relation itself), possibly at different sites. In ***horizontal fragmentation,*** each fragment consists of a subset of rows and original relation. In ***vertical fragmentation***, each fragment consists of subset of columns of the original relation. The following table illustrates the horizontal and vertical fragmentation

When a relation is fragmented, we must be able to recover the original relation from the fragments:

- **Horizontal fragmentation:** The union of the horizontal fragments must be equal to the original relation. Fragments are usually also required to be disjoint
- **Vertical fragmentation:** The collection of vertical fragments should be lossless join decomposition,

E_id	E_Name	City	Age	DoJ
100001	Narayani	Mumbai	42	12/06/1995
100002	Sasikala	Cochin	27	02/02/2000
100003	Reena	Chennai	22	03/12/2002
100004	Viji	Bangalore	30	17/03/2005
100005	Ambika	Chennai	27	19/11/2006
Vertical Fragmentation			**Horizontal Fragmentation**	

E_id	E_Name	City
100001	Narayani	Mumbai
100002	Sasikala	Cochin
100003	Reena	Chennai
100004	Viji	Bangalore
100005	Ambika	Chennai

Vertical Fragmentation

E_id	E_Name	City	Age	DoJ
100002	Sasikala	Cochin	27	02/02/2000
100003	Reena	Chennai	22	03/12/2002

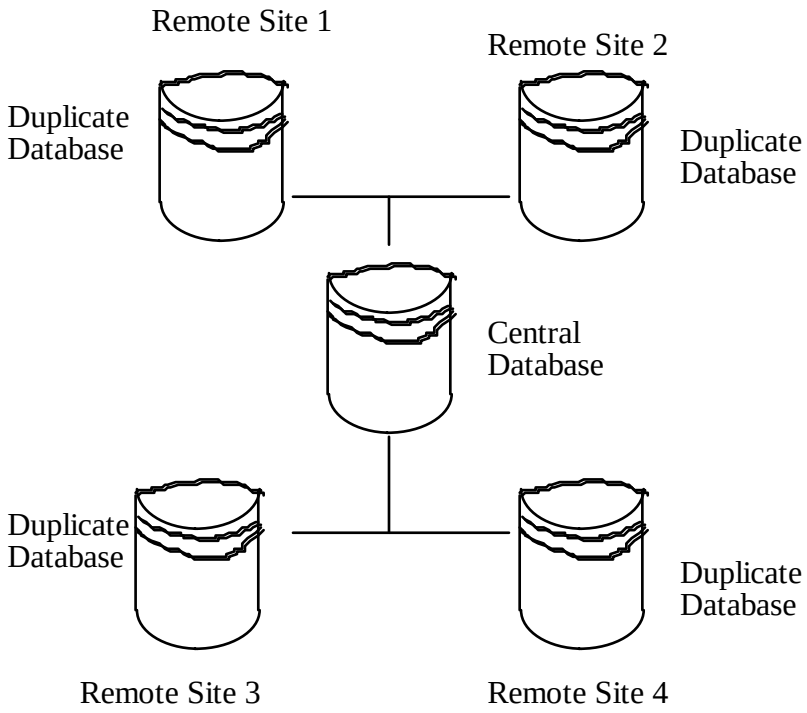
Horizontal Fragmentation

Hybrid Fragmentation

In most cases a simple horizontal or vertical fragmentation of a database schema will no be sufficient to satisfy the requirements of user applications. In this case a vertical fragmentation may be followed by a horizontal one, or vice versa, producing a tree-structured partitioning. Since the two types of partitioning strategies are applied one after the other, this alternative is called hybrid fragmentation. It has also been named mixed fragmentation or nested fragmentation.

9.7.2. Replication

Database replication is the frequent electronic copying data from a database in one computer or server to a database in another so that all users share the same level of information. The result is a distributed database in which users can access data relevant to their tasks without interfering with the work of others. The implementation of database replication for the purpose of eliminating data ambiguity or inconsistency among users is known as normalization.



REPLICATED DATABASE

With a **replicated database** a central database is duplicated at all other locations. This is the most appropriate for problems in which every location needs to access the same data. Another approach is to hold only one working copy of the data at the local node with each node storing the data that is most closely associated with it. The database is in fact distributed throughout the network. If a node needs access to records that are not held locally then this is obtained through the network - possibly by initially accessing a central index to find the location of the data. Software handles access to the database so that the fact that it is spread over a number of sites is not apparent to the user.

This approach requires more constant and heavier use of the network but it eliminates the problems of data redundancy and it removes the need for overnight reconciliation.

Database replication can be done in at least three different ways:

- Snapshot replication: Data on one server is simply copied to another server, or to another database on the same server.
- Merging replication: Data from two or more databases is combined into a single database.
- Transactional replication: Users receive full initial copies of the database and then receive periodic updates as data changes.

A distributed database management system (DDBMS) ensures that changes, additions, and deletions performed on the data at any given location are automatically reflected in the data stored at all the other locations. Therefore, every user always sees data that is consistent with the data seen by all the other users.

Advantages of Replication

There are five advantages of data replication:

Reliability: If one of the sites containing the relation (or database) fails, a copy can always be found at another site without network traffic delays. Also available copies can all be updated as soon as possible as transactions occur and unavailable nodes will be updated once they return to service.

Fast response: Each site that has a full copy can process queries locally, so queries can be processed rapidly.

Possible avoidance of complicated distributed transaction integrity routines: Replicated databases are usually refreshed at scheduled intervals, so most forms of replication are used when some relaxing of synchronization across database copies is acceptable.

Node decoupling: Each transaction may proceed without coordination across the network. Thus, if nodes are down, busy, or disconnected (e.g. in the case of mobile personal computers) a transaction is handled when the user desires. In the place of real-time synchronization of updates, a behind-the-scenes process coordinates all data copies.

Reduced network traffic at prime time: Often updating data happens during prime business hours, when network traffic is highest and the demands for rapid response greatest. Replication, with delayed updating of copies of data, moves network traffic for sending updates to other nodes to non-prime time hours.

Disadvantages of Replication

Storage requirements: Each site that has a full copy must have the same storage capacity that would be required if the data were stored centrally. Each copy requires storage space (the cost for which is constantly decreasing), and processing time is required to update each copy on each node.

Complexity and cost of updating: Whenever a relation is updated, it must (eventually) be updated at each site that holds a copy. Synchronizing updating near real time can require careful coordination, as will be clear later under the topic of commit protocol.

For these reasons, data replication is favored where most process request are read-only and where the data are relatively static, as in catalogs, telephone directories, train schedules, and so on.

9.8. LET US SUM UP

Local systems often store local databases but will still have access to a central online database. This central database can be accessed remotely or it can be held on local machines. A system is needed, therefore, to ensure that data integrity is maintained by ensuring the database is updated.
This means that computers have their own processing capabilities rather than all computers using a mainframe.

- Distributed systems may contain a number of separate but connected processors, although some centralized processing is possible.
- Files and programs need not be held centrally in a distributed system - functions can be passed onto regional centres e.g. Police
- Be aware of examples where data is distributed
- Be aware of examples where control within the database (different functions) are distributed
- Initially distributed systems still maintained a single central database. However the use of distributed data soon developed since otherwise a large amount of data needed to be moved through the network and a mainframe or telecommunications failure isolated the local processor from the data that it needed.
- A distributed database is a database that consists of two or more data files located at different sites on the network. Because the database is distributed, different users can access it without interfering with one another. However, the scattered data must be periodically synchronized to ensure data consistency.

9.9. LESSON END ACTIVITIES

1. List down various types of Distributed Databases with their applicatioins

9.10. POINTS FOR DISCUSSION

1. Discuss about various fragmentation techniques used in data distribution with respect to search engine.

9.11. REFERENCES http://en.wikipedia.org/wiki/Image:Wikibooks-logo-en.svg

1. Elmasri & Navathe, "Fundamentals of Database Systems, 3rd Edition, Pearson Education Asia, 2002
2. Stefans Ceri, Ginseppe Pelgatti "Distributed database Principles and systems" McGraw Hill

LESSON 10: DISTRIBUTED DBMS

CONTENTS

10.0 Aims and Objectives
10.1. Introduction
10.2. Distributed DBMS Architectures
10.3. Levels of Transparency
10.4. Distributed Database Design
10.5.Distributed DBMS Products
10.6. Features of Distributed File System
10.7. Overview of Concurrency Control And Recovery
10.8. Let us Sum UP
10.9. Lesson-End Activities
10.10. Points for Discussion
10.11. References

10.0. AIMS AND OBJECTIVES

After studying this lesson, you should be able to:

- Define the following key terms: distributed database, decentralized database, location transparency, synchronous distributed database, asynchronous distributed database, replication transparency, failure transparency, commits protocol, two-phase commit, and concurrency transparency.
- Explain the salient features of several distributed database management systems.

10.1. INTRODUCTION

To have a distributed database, there must be a database management system that coordinates the access to data at the various nodes. We will call such a system a distributed DBMS. Although each site may have a DBMS managing the local database at that site, a distributed DBMS is also required to perform the following functions:

- Keep track of where data are located in a distributed data dictionary.
- Determine the location from which to retrieve requested data and the location at which to process each part of a distributed query.
- If necessary, translate the request at one node using a local DBMS into the proper request to another node using a different DBMS and data model, and return data to the requesting node in the format accepted by that node.
- Provide data management functions such as security, concurrency and deadlock control, query optimization, and failure recovery.
- Provide consistency among copies of data across the remote sites.

10.2.DISTRIBUTED DBMS ARCHITECTURES

There are three alternative approaches to separating functionality across different DBMS-related processes; these alternative distributed DBMS architectures are called client-Server, Collaborating Server, and Middleware.

Client-Server Systems

A client-Server system has one or more client processes and one or more server processes, and a client process can send a query to any one server process. Clients are responsible for user-interface issues, and servers manage data and execute transactions. Thus, a client process could run on a personal computer and send queries to a server running a mainframe.

This architecture has become very popular for several reasons. First, it is relatively simple to implement due to its clean separation of functionality and because the server is centralized. Second, expensive server machines are not underutilized by dealing with mundane user-interactions, which are now relegated to inexpensive client machines. Their, uses can run a graphical user interface that they are familiar with, rather than the (possible unfamiliar and unfriendly) user interface on the server.

While writing Client-Server applications, it is important to remember the boundary between the client and the server and to keep the communication between them as set-oriented as possible. In particular, opening a cursor and fetching tuples one at a time generates many messages and should be avoided. (Even if we fetch several tuples and cache them at the client, messages must be exchanged when the cursor is advanced to ensure that the current row is locked.)

Collaborating Server System

The Client-Server architecture does not allow a single query to span multiple servers because the client process would have to be capable of breaking such a query into appropriate sub-queries to be executed at different sites and then piecing together the answers to the sub-queries. The client process would thus be quite complex, and its capabilities would begin to overlap with the server; distinguishing between clients and servers becomes harder. Eliminating this distinction leads us to alternatives to the Client-Server architecture: a Collaborating Server system. We can have a collection of database servers, each capable of running transactions against local data, which cooperatively execute transactions spanning multiple servers.

When a server receives a query that requires access to data at other servers, it generates appropriate sub queries to be executed by other servers and puts the results together to compute answers to the original query. Ideally, the decomposition of the query should be done using cost-based optimization, taking into account the cost of network communication as well as local processing cost.

Middleware Systems

The Middleware architecture is designed to allow a single query to span multiple servers, without requiring all database servers to be capable of managing such multiple site execution strategies. It is especially attractive when trying to integrate several legacy systems, whose basic capabilities cannot be extended.

The idea is that we need just one database server that is capable of managing queries and transactions spanning multiple servers; the remaining servers only need to handle local queries and transactions. We can think of this special server as a layer of software that coordinates the execution of queries and transactions across one or more independent database servers; such software is often called middleware. The middleware layers is capable of executing joins and other relational operations on data obtained from the other servers, but typically, does not itself maintain any data.

10.3. LEVELS OF TRANSPARENCY

A distributed DBMS should isolate users as much as possible from the complexities of distributed database management.

The classical view of a distributed database system is that the system should make the impact of data distribution transparent. In particular, the following properties are considered desirable:

Distributed data independence: Users should be able to ask queries without specifying where the referenced relations, or copies or fragments of the relations, are located. This principle is a natural extension of physical and logical data independence;

Distributed transaction atomicity: Users should be able to write transactions that access and update data at several sites just as they would write transactions over purely local data. In particular, the effects of a transaction across sites should continue to be atomic; that is, all changes persist if the transaction commits, and none persist if it aborts.

Stated differently, the distributed DBMS should make transparent the location of data in the network as well as other features of a distributed database. Four objectives of a distributed DBMS, when met, ease the construction of programs and the retrieval of data in distributed system these objectives, which are described below, are following: location transparency, replication transparency, failure transparency, and concurrency transparency

Location Transparency: Although data are geographically distributed and may move from place to place, with location transparency users (including programmers) can act as if all the data were located at a single node.

Replication Transparency: A design goal for a distributed database, which says that although a given data item may be replicated at several nodes in a network, a programmer or user may treat the data item as if it were a single item at a single node also called fragmentation transparency.

Failure transparency: A design goal for a distributed database, which guarantees that either all the actions of each transaction are committed or else none of them is committed.

Concurrency Transparency: A design goal for a distributed database, with the property that although a distributed system runs many transactions, it appears that a given transaction is the only activity in the system. Thus, when several transactions are

processed concurrently, the results must be the same as if each transaction were processed in serial order.

10.4. DISTRIBUTED DATABASE DESIGN

The basic steps to building a distributed database are similar to those for creating any database application. Once you identify the user needs, the developers organize the data through normalization, create queries using SQL, define the user interface, and build the application. However, the following table shows the steps designing a Distributed DBMS.

1. Design administration plan
2. Choose hardware and DBMS vendor, and network
3. Set up network and DBMS connections
4. Choose locations for data
5. Choose replication strategy
6. Create backup plan and strategy
7. Create local views and synonyms
8. Perform stress test: loads and failures

In particular, a network must connect the computers in all the locations. Even if the network already exists, it might have to be modified or extended to support the chosen hardware and DBMS software.

Another crucial step is determining where to store data. Backup and recovery plans are even more critical with a distributed database. Remember that several computers will be operating in different locations. Each system will probably have a different DBA. Yet the entire database must be protected from failures, so every system must have consistent backup and security plans. Developing these plans will probably require negotiation among the administrators- particularly when the systems cross national boundaries and multiple time zones. For example, it would be virtually impossible to backup data everywhere at the same time.

Once the individual system are installed and operational each location must create local views, synonym, and stored procedures that will connect the databases, grant access to the appropriate users, and connect the applications running on each system. Each individual link must be tested, and the final applications must be tested both for connections and for stress under heavy loads. It should also be tested for proper behavior when a network link is broken or a remote computer fails.

Operating and managing a distributed database system is considerably more difficult than handling a single database. Identifying the cause of problems is much more difficult. Basic tasks like backup and recovery require coordination of all DBAs/ some tools exist to make these jobs easier, but they can be improved.

Remember the rule that a distributed database should be transparent to the user. That same rule does not yet apply to DBAs or to application developers. Coordination

among administrators and developers is crucial to making applications more accessible to users.

10.5. DISTRIBUTED QUERY PROCESSING

The goal of distributed processing is to minimize the transfer of data on slower networks and to reduce the costs of network transfers. Part of this goal can be accomplished through design – developers must carefully choose where data should be located. Data should be stored as close as possible to where it will be used the most. However, trade-offs always arise when data is used in several locations.

Another issue in transferring data arises in terms of query processing. If a query needs to retrieve data from several different computers, the time to transfer the data and process the query depends heavily on how much data must be transferred and the speed of the transmission lines. Consequently, the result depends on how the DBMS joins the data from the multiple tables. In some cases the difference can be extreme. One method could produce a result in a few seconds. A different approach to the same query might take several days to process! Ideally, the DBMS should evaluate the query, the database used, and the transmission times to determine the most efficient way to answer the query.

10.5.1. Query Optimization

A good DBMS contains a query optimizer that checks the database contents and network transfer speeds to choose the best method to answer the query. You still might have to optimize some queries yourself. The basic rule is to transfer the least amount of data possible.

10.5. DISTRIBUTED DBMS PRODUCTS

Most of the leading vendors of database management systems have a distributed version. In most cases, to utilize all distributed database capabilities, one vendor's DBMS must be running at each node (a homogeneous distributed database environment). Client/server forms of a distributed database are arguable the most common form in existence today. Although these vendors' approaches are constantly changing, it is illustrative to overview how different vendors address distributed database management. Probably the most interesting aspects of the following table is the differences across the products.

Vendor	Product	Important features
IBM	DataPropagator Relational	Works with DB2 Primary site and asynchronous updates Read-only sites subscriber to primary site Subscription to subset or query result

	Distributed Relational Database Architecture (DRDA)	Heterogeneous database
Sybase	Replication Server	Middleware to access non-IBM databases Primary site and distributed read-only sites Update to read-only site as one transaction Hierarchical replication Data and stored procedures replicated
	SQL Anywhere	Mobile databases
	Omni SQL	Heterogeneous databases
Oracle	Table Snapshot Option	Periodic snapshots sent to read-only sites
	Symmetric Replication option	Asynchronous with multiple updateable copies Differential refresh DBA controls replication Two-phase commit
Computer Associates	CA-Ingres/Replicator	All database copies updateable Hierarchical replication DBA registers data for replication and other sites subscribe Master/slave form allows slave to be an Ingres database
	Ingres / Net and Ingres / Star	Decomposes query to distributed, homogeneous sites Two-phase commit Also used with non-Ingres databases
Microsoft	SQL Server	Primary site and distributed read-only sites Publish and subscribe, with articles and publications One database can pass copies of publications to other sites Mobile databases.

10.6. FEATURES OF DISTRIBUTED FILE SYSTEM

- **Transparent Management of Distributed and Replicated Data:** Transparency refers to separation of the higher-level semantics of a system form lower-level implementation issues. In other words, a transparent system "hides" the implementation details from users. The advantage of the fully transparent DBMS is the high level of support that it provides for the development of complex applications.

It is obvious that we would like to make all DBMS's (centralized of distributed) fully transparent.

- **Reliability Through Distributed Transactions:** Distributed files are intended to improve reliability since they have replicated components and, thereby eliminate single points of failure. The failure of as single site, or the failure of a communication link which makes one or more sites unreachable, is not sufficient to bring down the entire system. In the case of a distributed file system, this means that some of the data may be unreachable, but with proper care, users may be permitted to access other parts of the distributed database.
- **Improved Performance:** The case for the improved performance of distributed DBMSs is typically made based on two points:
  - A distributed DBMS fragments the conceptual database, enabling data to be stored in close proximity to its points of use (also called data localization).
  - The inherent parallelism of distributed systems may be exploited for inter-query parallelism. Inter-query parallelism results from the ability to execute multiple queries at the same time when intra-query parallelism is achieved by breaking up a single query into a number of sub-queries each of which is executed at a different site, accessing a different part of the distributed database.

10.7. OVERVIEW OF CONCURRENCY CONTROL AND RECOVERY

For concurrency control and recovery purposes, numerous problems arise in a distributed DBMS environment that is not encountered in a centralized DBMS environment. These include the following:

Dealing with multiple copies of data items: The concurrency control method is responsible for maintaining consistency among these copies. The recovery methods is responsible for making a copy consistent with other copies if the site on which the copy is stored fails and recovers later.
Failure of individual sites: The DDBMS should continue to operate with its running sites, if possible, when one or more individual sites fail. When a site recovers, its local database must be brought up to date with the rest of the sites before it rejoins the system.
Failure of communication links: The system must be able to deal with failure of one or more of the communication links that connect the sites. An extreme case of this problem is that network partitioning may occur. This breaks up the sites into two or more partitions, where the sites within each partition can communicate only with one another and not with sites in other partitions.
Distributed commit: Problems can arise with committing a transaction that is accessing databases stored on multiple sites if some sites fail during the commit process. The two-phase commit protocol is often used to deal with this problem.
Distributed deadlock: Deadlock may occur among several sites, so techniques for dealing with deadlocks must be extended to take this into account

Distributed concurrency control and recovery techniques must deal with these and other problems.

10.8. LET US SUM UP

In this lesson we provided an introduction to Distributed DBMS. Here we discussed the reasons for distribution and the potential advantages of distributed databases over centralized system; we also defined the concept of distribution transparency and the related concepts of fragmentation transparency and replication transparency. We discussed the design issues related to data fragmentation, replication, and distribution, and we distinguished between horizontal and vertical fragments of relations. We discussed the use of data replication to improve system reliability and availability. We categorized DDBMS by using criteria such as degree of homogeneity of software modules and degree of local autonomy. We discussed the issues of federated database management in some detail focusing on the needs of supporting various types of autonomies and dealing with semantic heterogeneity.

10.9. LESSON END ACTIVITIES

1. What additional functions does a DDBMS have over a centralized DBMS?
2. What are the main software modules of a DDBMS? Discuss the main functions of each of these modules in the context of the client-server architecture.

10.10. POINTS FOR DISCUSSION

Discuss what is meant by the following terms:

- Degree of homogeneity of a DDBMS
- Degree of local autonomy of a DDBMS
- Federated DBMS,
- Distribution transparency
- Fragmentation transparency,
- Replication transparency
- Mulidatabase system

10.11. REFERENCES http://en.wikipedia.org/wiki/Image:Wikibooks-logo-en.svg

3. Elmasri & Navathe, “Fundamentals of Database Systems, 3rd Edition, Pearson Education Asia, 2002.
4. Stefans Ceri, Ginseppe Pelgatti “Distributed database Principles and systems” McGraw Hill.