NEHRU COLLEGE OF ENGINEERING AND RESEARCH CENTRE
(NAAC Accredited)
(Approved by AICTE, Affiliated to APJ Abdul Kalam Technological University, Kerala)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE MATERIALS

CS407 DISTRIBUTED COMPUTING

VISION OF THE INSTITUTION

To mould true citizens who are millennium leaders and catalysts of change through excellence in education.

MISSION OF THE INSTITUTION

NCERC is committed to transform itself into a center of excellence in Learning and Research in Engineering and Frontier Technology and to impart quality education to mould technically competent citizens with moral integrity, social commitment and ethical values.

We intend to facilitate our students to assimilate the latest technological know-how and to imbibe discipline, culture and spiritually, and to mould them in to technological giants, dedicated research scientists and intellectual leaders of the country who can spread the beams of light and happiness among the poor and the underprivileged.

ABOUT DEPARTMENT

- ♦ Established in: 2002
- ♦ Course offered : B.Tech in Computer Science and Engineering

  M.Tech in Computer Science and Engineering

  M.Tech in Cyber Security
- ♦ Approved by AICTE New Delhi and Accredited by NAAC
- ♦ Affiliated to the University of A P J Abdul Kalam Technological University.

DEPARTMENT VISION

Producing Highly Competent, Innovative and Ethical Computer Science and Engineering Professionals to facilitate continuous technological advancement.

DEPARTMENT MISSION

1. To Impart Quality Education by creative Teaching Learning Process
2. To Promote cutting-edge Research and Development Process to solve real world problems with emerging technologies.
3. To Inculcate Entrepreneurship Skills among Students.
4. To cultivate Moral and Ethical Values in their Profession.

PROGRAMME EDUCATIONAL OBJECTIVES

PEO1: Graduates will be able to Work and Contribute in the domains of Computer Science and Engineering through lifelong learning.

PEO2: Graduates will be able to Analyse, design and development of novel Software Packages, Web Services, System Tools and Components as per needs and specifications.

PEO3: Graduates will be able to demonstrate their ability to adapt to a rapidly changing environment by learning and applying new technologies.

PEO4: Graduates will be able to adopt ethical attitudes, exhibit effective communication skills, Teamworkand leadership qualities.

PROGRAM OUTCOMES (POS)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSO)

PSO1: Ability to Formulate and Simulate Innovative Ideas to provide software solutions for Real-time Problems and to investigate for its future scope.

PSO2: Ability to learn and apply various methodologies for facilitating development of high quality System Software Tools and Efficient Web Design Models with a focus on performance optimization.

PSO3: Ability to inculcate the Knowledge for developing Codes and integrating hardware/software products in the domains of Big Data Analytics, Web Applications and Mobile Apps to create innovative career path and for the socially relevant issues.

COURSE OUTCOMES

CO1	Distinguish distributed computing paradigm from other computing paradigms
CO2	Identify the core concepts of distributed systems
CO3	Illustrate the mechanisms of inter process communication in distributed system
CO4	Apply appropriate distributed system principles in ensuring transparency, consistency and fault-tolerance in distributed file system
CO5	Compare concurrency control mechanisms in distributed transaction environment
CO6	Demonstrate the need for Mutual exclusion and election algorithms

MAPPING OF COURSE OUTCOMES WITH PROGRAM OUTCOMES

	PO 1	**PO 2**	**PO 3**	**PO 4**	**PO 5**	**PO 6**	**PO 7**	**PO 8**	**PO 9**	**PO 10**	**PO 11**	**PO 12**
CO1	3											
CO2	3											
CO3	2	3										
CO4			3									
CO5		3	3									
CO6		2	2									

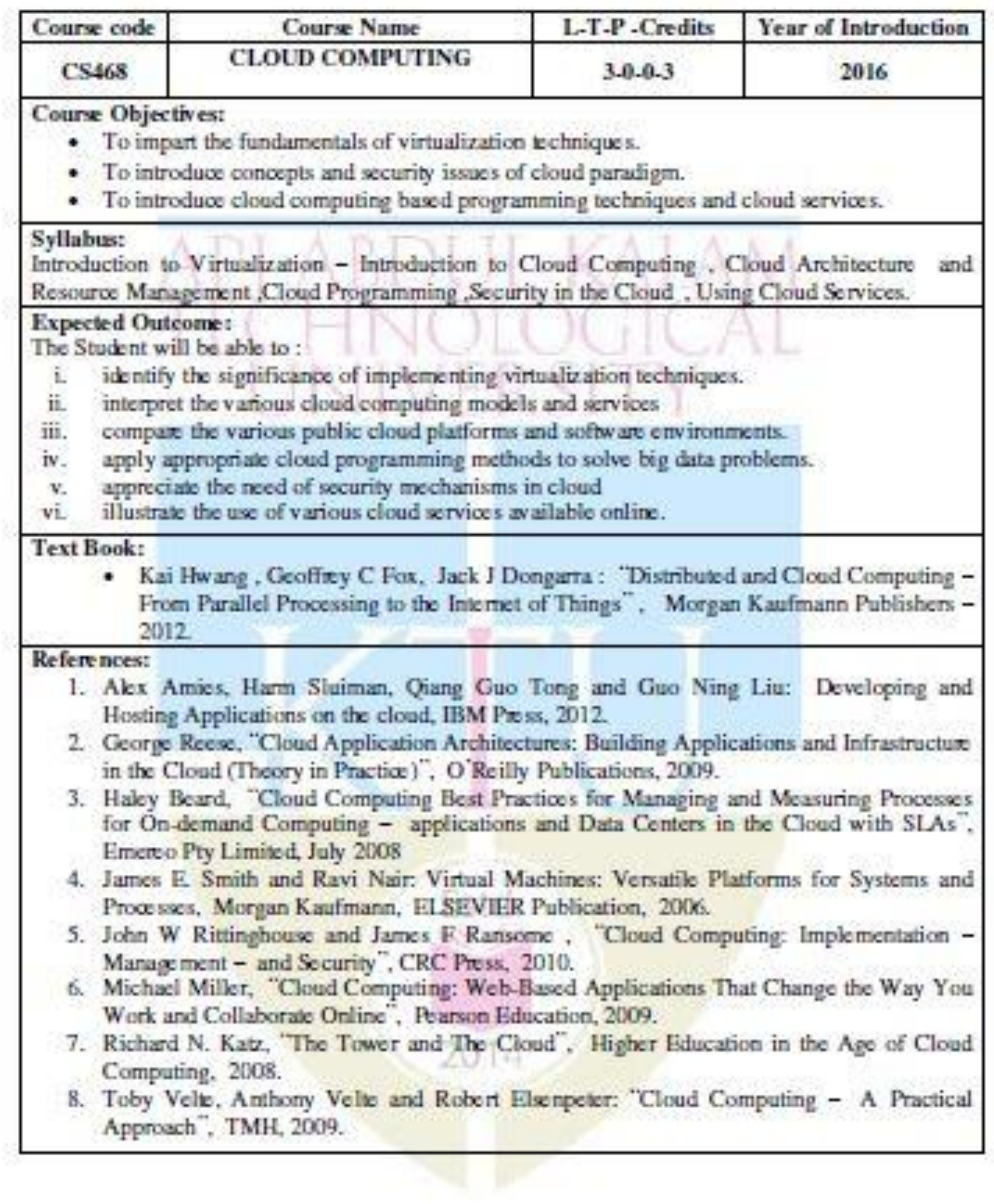
Note: H-Highly correlated=3, M-Medium correlated=2, L-Less correlated=1

MAPPING OF COURSE OUTCOMES WITH PROGRAM SPECIFIC OUTCOMES

	PSO 1	**PSO 2**	**PSO 3**
CO1	3		
CO2	2		
CO3		3	
CO4		3	
CO5	3		
CO6		3	

Note: H-Highly correlated=3, M-Medium correlated=2, L-Less correlated=1

SYLLABUS



Course code	Course Name	L-T-P -Credits	Year of Introduction
CS468	CLOUD COMPUTING	3-0-0-3	2016

Course Objectives:

- To impart the fundamentals of virtualization techniques.
- To introduce concepts and security issues of cloud paradigm.
- To introduce cloud computing based programming techniques and cloud services.

Syllabus:
Introduction to Virtualization – Introduction to Cloud Computing , Cloud Architecture and Resource Management ,Cloud Programming ,Security in the Cloud , Using Cloud Services.

Expected Outcome:
The Student will be able to :

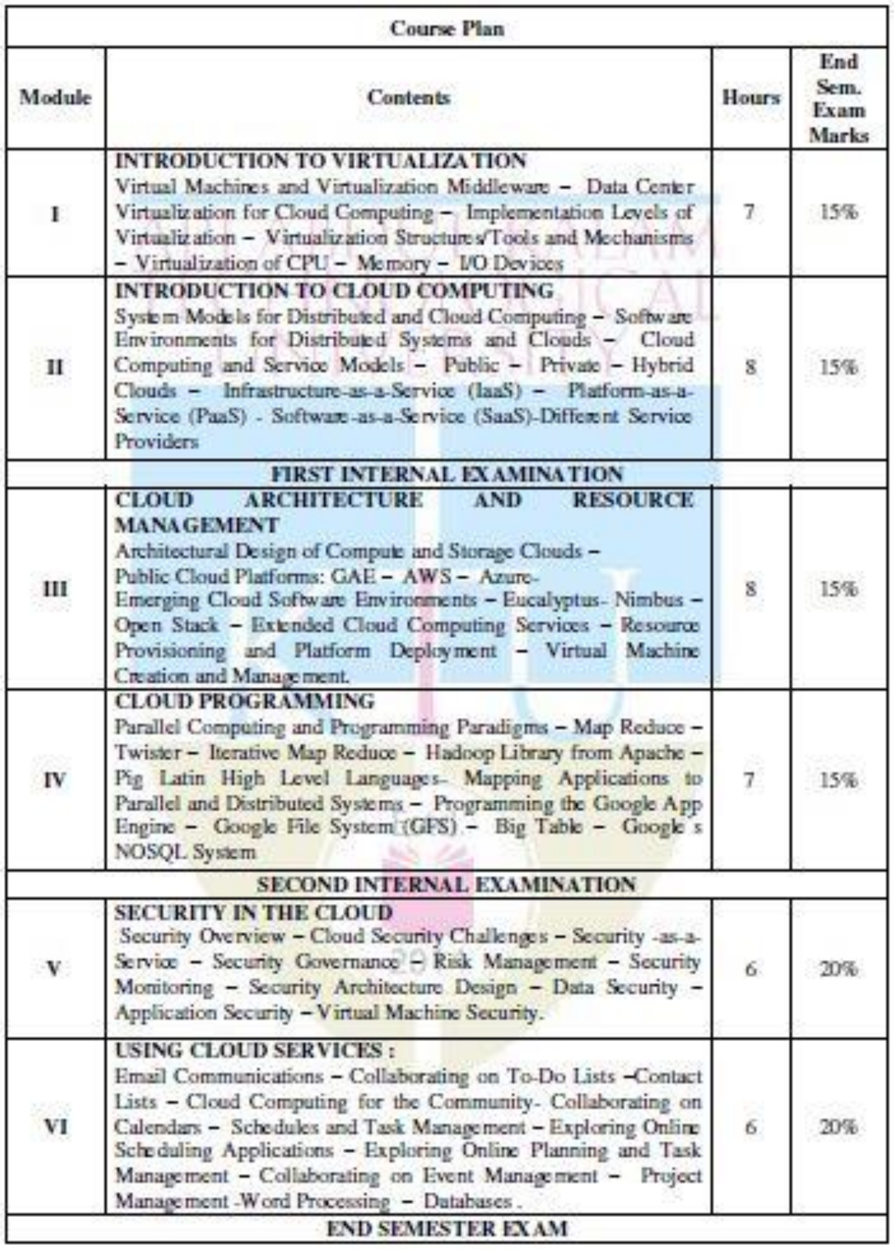
i. identify the significance of implementing virtualization techniques.
ii. interpret the various cloud computing models and services
iii. compare the various public cloud platforms and software environments.
iv. apply appropriate cloud programming methods to solve big data problems.
v. appreciate the need of security mechanisms in cloud
vi. illustrate the use of various cloud services available online.

Text Book:

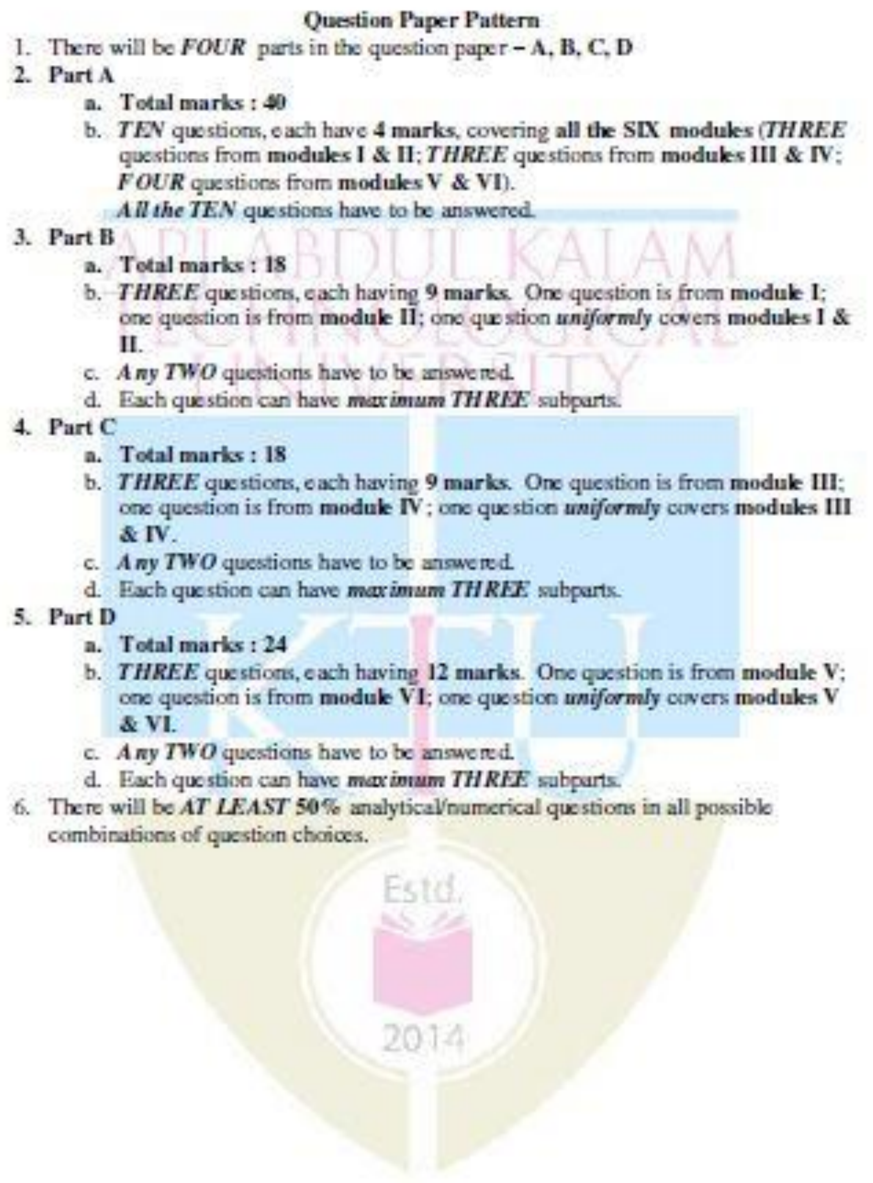
- Kai Hwang , Geoffrey C Fox, Jack J Dongarra : "Distributed and Cloud Computing – From Parallel Processing to the Internet of Things" , Morgan Kaufmann Publishers – 2012.

References:

1. Alex Amies, Harm Sluiman, Qiang Guo Tong and Guo Ning Liu: Developing and Hosting Applications on the cloud, IBM Press, 2012.
2. George Reese, "Cloud Application Architectures: Building Applications and Infrastructure in the Cloud (Theory in Practice)", O'Reilly Publications, 2009.
3. Haley Beard, "Cloud Computing Best Practices for Managing and Measuring Processes for On-demand Computing – applications and Data Centers in the Cloud with SLAs", Emereo Pty Limited, July 2008
4. James E. Smith and Ravi Nair: Virtual Machines: Versatile Platforms for Systems and Processes, Morgan Kaufmann, ELSEVIER Publication, 2006.
5. John W Rittinghouse and James F Ransome , "Cloud Computing: Implementation – Management – and Security", CRC Press, 2010.
6. Michael Miller, "Cloud Computing: Web-Based Applications That Change the Way You Work and Collaborate Online", Pearson Education, 2009.
7. Richard N. Katz, "The Tower and The Cloud", Higher Education in the Age of Cloud Computing, 2008.
8. Toby Velte, Anthony Velte and Robert Elsenpeter: "Cloud Computing – A Practical Approach", TMH, 2009.



Course Plan			
Module	**Contents**	**Hours**	**End Sem. Exam Marks**
I	**INTRODUCTION TO VIRTUALIZATION** Virtual Machines and Virtualization Middleware – Data Center Virtualization for Cloud Computing – Implementation Levels of Virtualization – Virtualization Structures/Tools and Mechanisms – Virtualization of CPU – Memory – I/O Devices	7	15%
II	**INTRODUCTION TO CLOUD COMPUTING** System Models for Distributed and Cloud Computing – Software Environments for Distributed Systems and Clouds – Cloud Computing and Service Models – Public – Private – Hybrid Clouds – Infrastructure-as-a-Service (IaaS) – Platform-as-a-Service (PaaS) - Software-as-a-Service (SaaS)-Different Service Providers	8	15%
FIRST INTERNAL EXAMINATION			
III	**CLOUD ARCHITECTURE AND RESOURCE MANAGEMENT** Architectural Design of Compute and Storage Clouds – Public Cloud Platforms: GAE – AWS – Azure- Emerging Cloud Software Environments – Eucalyptus- Nimbus – Open Stack – Extended Cloud Computing Services – Resource Provisioning and Platform Deployment – Virtual Machine Creation and Management.	8	15%
IV	**CLOUD PROGRAMMING** Parallel Computing and Programming Paradigms – Map Reduce – Twister – Iterative Map Reduce – Hadoop Library from Apache – Pig Latin High Level Languages- Mapping Applications to Parallel and Distributed Systems – Programming the Google App Engine – Google File System (GFS) – Big Table – Google s NOSQL System	7	15%
SECOND INTERNAL EXAMINATION			
V	**SECURITY IN THE CLOUD** Security Overview – Cloud Security Challenges – Security -as-a-Service – Security Governance – Risk Management – Security Monitoring – Security Architecture Design – Data Security – Application Security – Virtual Machine Security.	6	20%
VI	**USING CLOUD SERVICES :** Email Communications – Collaborating on To-Do Lists –Contact Lists – Cloud Computing for the Community- Collaborating on Calendars – Schedules and Task Management – Exploring Online Scheduling Applications – Exploring Online Planning and Task Management – Collaborating on Event Management – Project Management -Word Processing – Databases .	6	20%
END SEMESTER EXAM			



Question Paper Pattern

1. There will be ***FOUR*** parts in the question paper **– A, B, C, D**
2. **Part A**
   a. **Total marks : 40**
   b. ***TEN*** questions, each have **4 marks**, covering **all the SIX modules** (***THREE*** questions from **modules I & II**; ***THREE*** questions from **modules III & IV**; ***FOUR*** questions from **modules V & VI**).
   All the TEN questions have to be answered.
3. **Part B**
   a. **Total marks : 18**
   b. ***THREE*** questions, each having **9 marks**. One question is from **module I**; one question is from **module II**; one question ***uniformly*** covers **modules I & II**.
   c. ***Any TWO*** questions have to be answered.
   d. Each question can have ***maximum THREE*** subparts.
4. **Part C**
   a. **Total marks : 18**
   b. ***THREE*** questions, each having **9 marks**. One question is from **module III**; one question is from **module IV**; one question ***uniformly*** covers **modules III & IV**.
   c. ***Any TWO*** questions have to be answered.
   d. Each question can have ***maximum THREE*** subparts.
5. **Part D**
   a. **Total marks : 24**
   b. ***THREE*** questions, each having **12 marks**. One question is from **module V**; one question is from **module VI**; one question ***uniformly*** covers **modules V & VI**.
   c. ***Any TWO*** questions have to be answered.
   d. Each question can have ***maximum THREE*** subparts.
6. There will be ***AT LEAST* 50%** analytical/numerical questions in all possible combinations of question choices.

QUESTION BANK

MODULE I				
Q:NO:	**QUESTIONS**	**CO**	**KL**	**PAGE NO:**
1	Illustrate the characteristics of distributed systems.	CO1	K3	14
2	Point out the examples of distributed systems.	CO1	K4	14
3	Explain about issues in designing distributed systems	CO1	K5	15
4	Describe the different failure handling methods.	CO1	K2	18
5	List out the different transparencies.	CO1	K4	19
6	Explain about mini computer model.	CO1	K2	20
7	Differentiate between workstation and workstation-server model.	CO1	K4	21,22
8	Compare and contrast between processor pool and hybrid model.	CO1	K4	24,25
9	Write note on trends in distributed systems.	CO1	K6	26
MODULE II				
1	Explain briefly about architectural model.	CO2	K5	28
2	List out the different communication entities.	CO2	K4	29
3	Describe about communication paradigm.	CO2	K2	30
4	Demonstrate the variations in client server model with example.	CO2	K3	35
5	Explain about software layers.	CO2	K2	37

6	Write note on interaction model.	CO2	K6	41
7	Explain the different failures in failure model.	CO2	K2	44
8	Describe about the three generations of physical model.	CO2	K2	48
MODULE III				
1	Point out the characteristics of inter process communication.	CO3	K4	51
2	Differentiate between TCP and UDP protocols.	CO3	K4	53,55
3	Explain briefly about group communication	CO3	K2	57
4	Briefly describe about types of groups.	CO3	K2	59
5	Write short note on group membership management.	CO3	K3	61
6	Explain about IP multicast.	CO3	K2	65
7	Describe about RPC call semantics.	CO3	K2	68
8	Explain about implementation of RPC.	CO3	K5	69
9	Define network virtualization.	CO3	K1	71
10	Write short note on Skype architecture.	CO3	K3	72
MODULE IV				
1	Write briefly about distributed file systems.	CO4	K6	75
2	Explain about DFS modules.	CO4	K2	75
3	Point out the requirements in DFS.	CO4	K4	76
4	Explain about file system architecture.	CO4	K2	77
5	Describe briefly about Sun NFS.	CO4	K2	80

6	Write short note on name services.	CO4	K3	86
7	Explain about Domain Name Service.	CO4	K2	90
8	Briefly explain about name servers and navigation.	CO4	K5	92
MODULE V				
1	Explain about transactions in detail.	CO5	K5	100
2	Point out the ACID properties.	CO5	K4	100
3	Distinguish between lost update and dirty read problem.	CO5	K4	103, 104
4	Explain about conflicting operations.	CO5	K2	106
5	Write short note on nested transactions.	CO5	K3	108
6	Compare and contrast between two phase and strict two phase locking methods.	CO5	K4	109,110
7	Write notes on deadlocks.	CO5	K6	112
8	Explain briefly about optimistic concurrency control.	CO5	K5	113
MODULE VI				
1	Briefly describe about essential requirement for mutual exclusion.	CO6	K2	116
2	Explain about central server algorithm.	CO6	K2	117
3	Distinguish between ring based and Ricart Agarwal algorithm.	CO6	K4	118, 119
4	Explain about Maekawa’s voting algorithm.	CO6	K5	121
5	Write short note on elections.	CO6	K3	122
6	Explain about ring based election algorithm.	CO6	K5	123
7	Write note on Bully algorithm.	CO6	K6	124

APPENDIX 1

CONTENT BEYOND THE SYLLABUS

Module 1

Evolution of Distributed Computing -Issues in designing a distributed system-Challenges- Minicomputer model –Workstation model - Workstation-Server model–Processor - pool model- Trends in distributed systems

1.1 Evolution of Distributed Computing

- At the very beginning, one computer could only do one particular task at a time (batch processing). Then multiprogramming introduced, which can execute multiple programs but there is only one processor, there can be no true simultaneous execution of different programs.
- If we need multiple tasks to be done in parallel, we need to have multiple computers running in parallel. Multiprocessor run multiple programs simultaneously by sharing same memory, so it is tightly coupled system.
- But running them parallel was not enough for building a truly distributed system since it requires a mechanism to communicate between different computers.
- Distributed Computing is a field of computer science that studies distributed systems.
- A distributed system is a model in which components located on networked computers communicate and coordinate their actions by passing messages. Distributed system is said to be loosely coupled because each processor has its own local memory.

Characteristics of Distributed System

● Concurrency: Tasks carry out independently

● No global clock : Each system has its own clock and it coordinate their actions by exchanging messages

● Independent failures : when some systems fail, others may not know, does not stop the running of the whole system

Example Of Distributed Systems

1. The Internet
2. Mobile Computing
3. Intranet
4. Multiplayer online game

1. The Internet: A vast interconnected collection of computer networks of many different types. Programs running on the computers connected to it interact by passing messages, employing a common means of communication

2. Mobile Computing: Mobile computing (also called nomadic computing) is the performance of computing tasks while the user is on the move, or visiting places other than their usual environment.

3. Intranet: A portion of the Internet that is separately administered and has a boundary that can be configured to enforce local security policies and composed of several LANs linked by backbone connections.

1.2 Issues in designing a distributed system/ Challenges

In distributed system, there is no global clock among the multiple processor. So it is very hard to schedule the processor. Designing a distributed system does not come as easy and straight forward. A number of challenges need to be overcome in order to get the ideal system. The major challenges in distributed systems are listed below:

● Heterogeneity

● Openness

● Security

● Scalability

● Failure handling

● Concurrency

● Transparency

Heterogeneity:

The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks. Heterogeneity (that is, variety and difference) applies to all of the following:

• Networks;

• Computer hardware;

• Operating systems;

• Programming languages;

• Implementations by different developers.

Different programming languages use different representations for characters and data structures such as arrays and records. These differences must be addressed if programs written in different languages are to be able to communicate with one another.

Programs written by different developers cannot communicate with one another unless they use common standards.

- Middleware: The term middleware applies to a software layer that provides a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware, operating systems and programming languages.

- Heterogeneity and mobile code: The term mobile code is used to refer to program code that can be transferred from one computer to another and run at the destination – Java applets are an example. Code suitable for running on one computer is not necessarily suitable for running on another because executable programs are normally specific both to the instruction set and to the host operating system.

Openness:

The openness of a computer system is the characteristic that determines whether the system can be extended and re-implemented in various ways.

The openness of distributed systems is determined primarily by the degree to which new resource-sharing services can be added and be made available for use by a variety of client programs.

To summarize:

• Open systems are characterized by the fact that their key interfaces are published.

• Open distributed systems are based on the provision of a uniform communication mechanism and published interfaces for access to shared resources.

• Open distributed systems can be constructed from heterogeneous hardware and software, possibly from different vendors

Security:

Security for information resources has three components:

-confidentiality (protection against disclosure to unauthorized individuals)

-integrity (protection against alteration or corruption)

-availability (protection against interference with the means to access the resources).

However, security challenges have not yet been fully met: for Denial of service attacks' and Security of mobile code'.

Scalability:

Distributed systems operate effectively and efficiently at many different scales, ranging from a small intranet to the Internet.

A system is described as scalable if it will remain effective when there is a significant increase in the number of resources and the number of users.

The number of computers and servers in the Internet has increased dramatically.

The design of scalable distributed systems presents the following challenges:

● controlling the cost of physical resources

● controlling the performance loss

● Preventing software resources running out

● Avoiding performance bottlenecks

Failure handling

Computer systems sometimes fail. When faults occur in hardware or software, programs may produce incorrect results or may stop before they have completed the intended computation.

Failures in a distributed system are partial – that is, some components fail while others continue to function. Therefore the handling of failures is particularly difficult.

● Detecting failures: Some failures can be detected. For example, checksums can be used to detect corrupted data in a message or a file

● Masking failures: Some failures that have been detected can be hidden or made less severe. Two examples of hiding failures:

1. Messages can be retransmitted when they fail to arrive.

2. File data can be written to a pair of disks so that if one is corrupted, the other may still be correct.

● Tolerating failures: Most of the services in the Internet do exhibit failures – it would not be practical for them to attempt to detect and hide all of the failures that might occur in such a large network with so many components. When a web browser cannot contact a web server, it does not make the user wait forever while it keeps on trying – it informs the user about the problem, leaving them free to try again later.

● Recovery from failures: Recovery involves the design of software so that the state of permanent data can be recovered or ‘rolled back’ after a server has crashed

Concurrency

Both services and applications provide resources that can be shared by clients in a distributed system. There is therefore a possibility that several clients will attempt to access a shared resource at the same time

Transparency

Transparency is defined as the concealment from the user and the application programmer of the separation of components in a distributed system, so that the system is perceived as a whole rather than as a collection of independent components.

● Access transparency: enables local and remote resources to be accessed using identical operations.

● Location transparency: enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address).

● Concurrency transparency: enables several processes to operate concurrently using shared resources without interference between them.

● Replication transparency: enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.

● Failure transparency: enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.

● Mobility transparency: allows the movement of resources and clients within a system without affecting the operation of users or programs.

● Performance transparency: allows the system to be reconfigured to improve performance as loads vary.

The most important transparency are access and location transparency. They are referred together as network transparency.

1.3 Distributed computing system models

1.3.1 Minicomputer Model

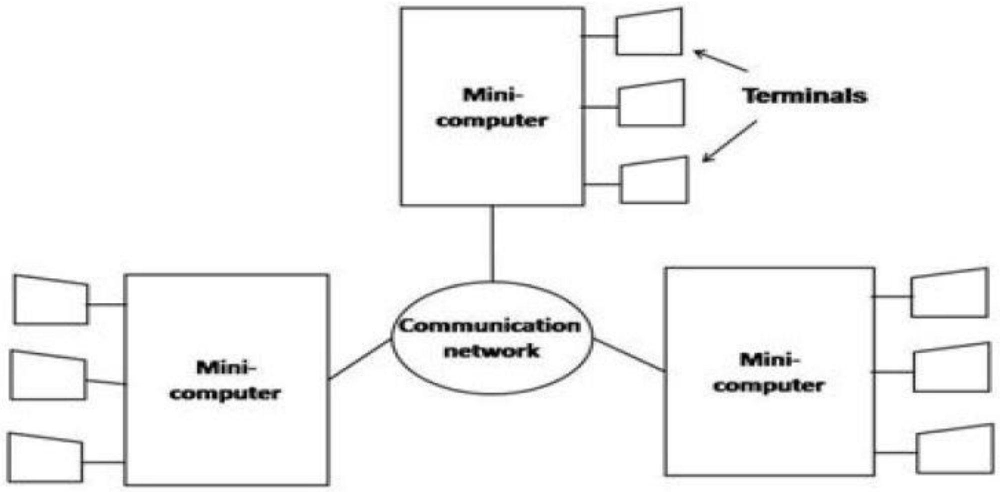
● The minicomputer model is a simple extension of the centralized time-sharing system.

● A distributed computing system based on this model consists of a few minicomputers interconnected by a communication network were each minicomputer usually has multiple users simultaneously logged on to it.

● Several interactive terminals are connected to each minicomputer. Each user logged on to one specific minicomputer has remote access to other minicomputers.

● The network allows a user to access remote resources that are available on some machine other than the one on to which the user is currently logged. The minicomputer model may be used when resource sharing with remote users is desired.

● The early ARPA net is an example of a distributed computing system based on the minicomputer model.



1.3.2 Workstation Model

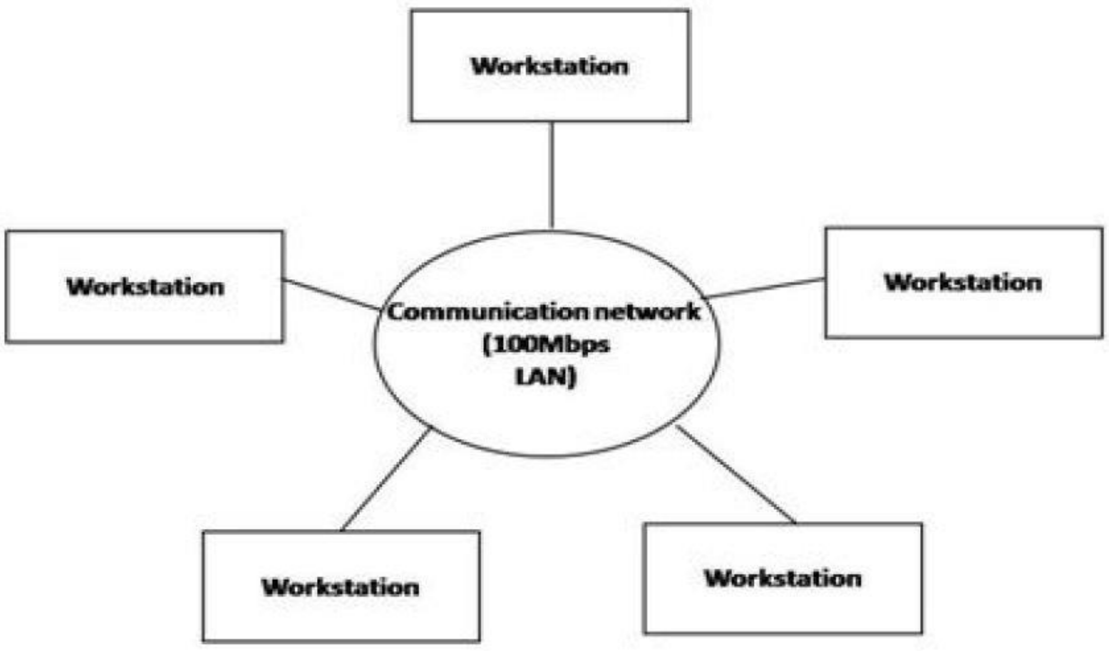
● A distributed computing system based on the workstation model consists of several workstations interconnected by a communication network.

● An organization may have several workstations located throughout an infrastructure were each workstation is equipped with its own disk & serves as a single-user computer.

● In such an environment, at any one time a significant proportion of the workstations are idle which results 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 & do not have sufficient processing power at their own workstations to get their jobs processed efficiently.

● Example: Sprite system & Xerox PARC.



1.3.3 Workstation–Server Model

● The workstation model is a network of personal workstations having its own disk & a local file system.

● A workstation with its own local disk is usually called a diskful workstation & a workstation without a local disk is called a diskless workstation. 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 & several workstations interconnected by a communication network.

● In this model, a user logs onto a workstation called his or her home workstation .Normal computation activities required by the user's processes are performed at the user's home workstation, but requests for services provided by special servers are sent to a server providing that type of service that performs the user's requested activity & returns the result of request processing to the user's workstation.

● Therefore, in this model, the user's processes need not migrated to the server machines for getting the work done by those machines.

● Example: The V-System.

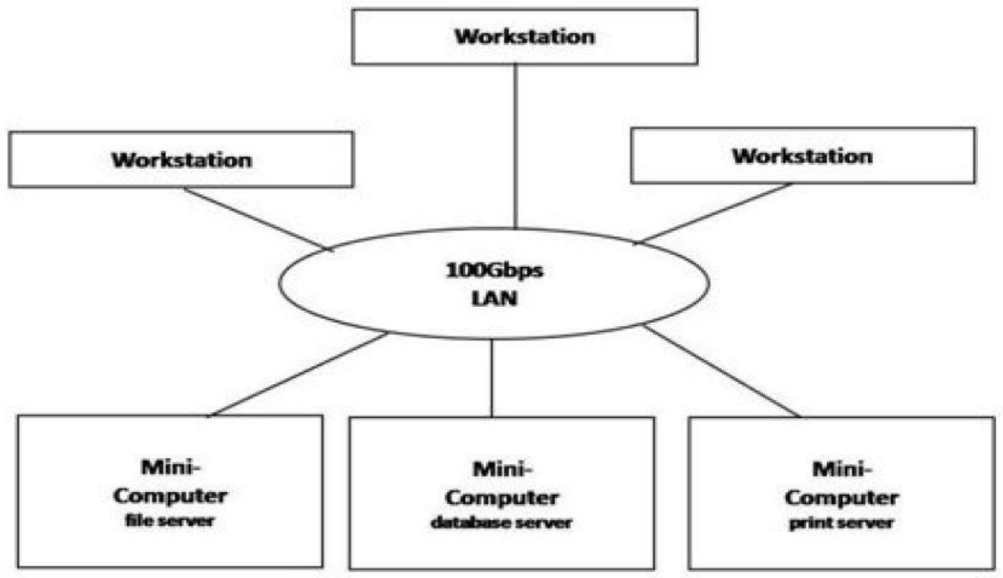
Advantages:

1. User has guaranteed response time

2. Does not need process migration facility due to client-server mode l of communication

3. Users have flexibility to use any workstation and access any files.

4. Backup and hardware maintenance are easier with diskless workstation.



1.3.4 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 the user may need a very large amount of computing power for a short time.

● Therefore, unlike the workstation-server model in which a processor is allocated to each user, in processor-pool model the processors are pooled together to be shared by the users as needed.

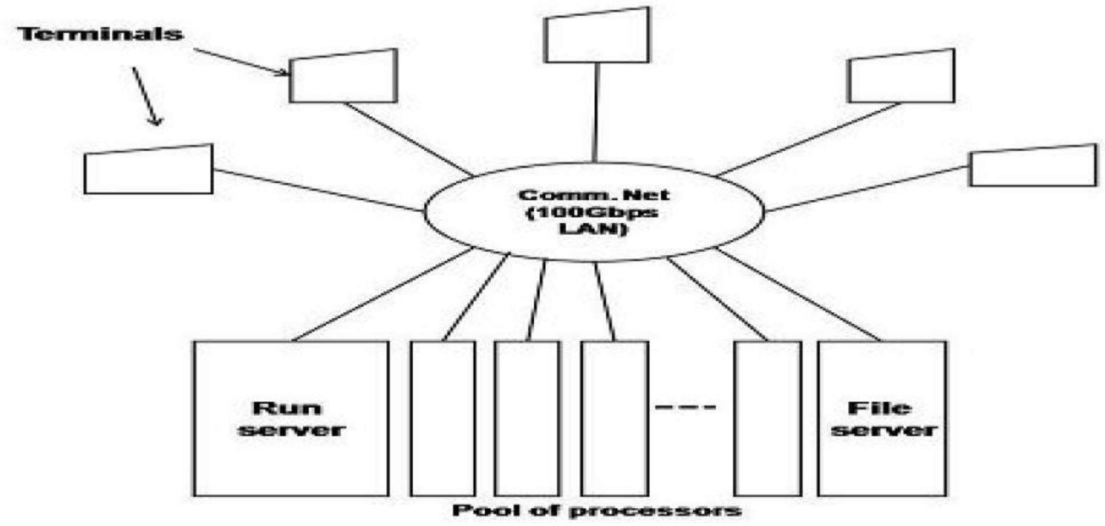
● The pool of processors consists of a large number of microcomputers & minicomputers attached to the network.

● Each processor in the pool has its own memory to load & run a system program or an application program of the distributed computing system.

● In this model no home machine is present & the user does not log onto any machine.

● This model has better utilization of processing power & greater flexibility.

● Example: Amoeba & the Cambridge Distributed Computing System.



1.3.5 Hybrid Model

● The workstation-server model has a large number of computer users only performing simple interactive tasks &-executing small programs.

● In a working environment that has groups of users who often perform jobs needing massive computation, the processor-pool model is more attractive & suitable.

● To combine Advantages of workstation-server & processor-pool models, a hybrid model can be used to build a distributed system.

● The processors in the pool can be allocated dynamically for computations that are too large or require several computers for execution.

● The hybrid model gives guaranteed response to interactive jobs allowing them to be more processed in local workstations of the users

1.4 Trends in distributed systems

Distributed systems are undergoing a period of significant change and this can be traced back to a number of influential trends:

• The emergence of pervasive networking technology

• The emergence of ubiquitous computing coupled with the desire to support user mobility in distributed systems

• The increasing demand for multimedia services

• The view of distributed systems as a utility.

1.4.1 Pervasive networking and the modern Internet

The modern Internet is a vast interconnected collection of computer networks of many different types, with the range of types increasing all the time. Example, a wide range of wireless communication technologies such as WiFi, WiMAX, Bluetooth and third-generation mobile phone networks. The net result is that networking has become a pervasive resource and devices can be connected (if desired) at any time and in any place

1.4.2 Mobile and ubiquitous computing

Technological advances in device miniaturization and wireless networking have led increasingly to the integration of small and portable computing devices into distributed systems. These devices include:

● Laptop computers.

● Handheld devices, including mobile phones, smart phones, GPS-enabled devices, pagers, personal digital assistants (PDAs), video cameras and digital cameras.

● Wearable devices, such as smart watches with functionality similar to a PDA.

● Devices embedded in appliances such as washing machines, hi-fi systems, cars and refrigerators.

The portability of many of these devices, together with their ability to connect conveniently to networks in different places, makes mobile computing possible.

Mobile computing is the performance of computing tasks while the user is on the move, or visiting places other than their usual environment.

1.4.3 Distributed multimedia systems

Another important trend is the requirement to support multimedia services in distributed systems. The benefits of distributed multimedia computing are considerable in that a wide range of new (multimedia) services and applications can be provided on the desktop, including access to live or pre-recorded television broadcasts, access to film libraries offering video-on-demand services, access to music libraries, the provision of audio and video conferencing facilities and integrated telephony features including IP telephony or related technologies such as Skype, a peer-to-peer alternative to IP telephony.

Webcasting is an application of distributed multimedia technology. Webcasting is the ability to broadcast continuous media, typically audio or video, over the Internet.

1.4.4 Distributed computing as a utility

With the increasing maturity of distributed systems infrastructure, a number of companies are promoting the view of distributed resources as a commodity or utility, drawing the analogy between distributed resources.

eg. Cloud computing, grid computing

Module 2

System models: Physical models - Architectural models -Fundamental models

System Models:

It describes **common properties** and **design choice of dispatcher** for distributed system in a single descriptive model.

Three types of models

● **Architectural Models**

● **Fundamental Models**

● **Physical Models**

2.1 Architectural Models:

Architecture models define the main components of the system, what their roles are and how they interact (software architecture), and how they are deployed in an underlying network of computers (system architecture).

Architecture model is concerned with the placement of its parts, namely how components are mapped to underlying network and the relationship between them, that is, their functional roles and patterns of communication between them.

Architectural Model- including,

● System Architectures

● Architectural elements

● Software layers

● Variations on the client-server model

2.1.1 Architectural elements

To understand the fundamental building blocks of a distributed system, it is necessary to consider four key questions:

• What are the entities that are communicating in the distributed system?

• How do they communicate, or, more specifically, what communication paradigm is used?

• What (potentially changing) roles and responsibilities do they have in the overall architecture?

• How are they mapped on to the physical distributed infrastructure (what is their placement)?

Communicating entities: what is communicating and how those entities communicate together define a rich design space for the distributed systems developer to consider. It is helpful to address the first question from a system-oriented and a problem-oriented perspective.

From a system perspective, the answer is normally very clear in that the entities that communicate in a distributed system are typically processes, leading to the prevailing view of a distributed system as processes coupled with appropriate inter-process communication paradigms.

From a programming perspective, however, this is not enough, and moreproblem-oriented abstractions have been proposed:

Objects: Objects have been introduced to enable and encourage the use of object oriented approaches in distributed systems (including both object-oriented design and object-oriented programming languages).

Components: Since their introduction a number of significant problems have been identified with distributed objects, and the use of component technology has emerged as a direct response to such weaknesses. Components resemble objects in that they offer problem-oriented abstractions for building distributed systems and are also accessed through interfaces.

Web services : Web services represent the third important paradigm for the development of distributed systems. Web services are closely related to objects and components, again taking an approach based on encapsulation of behavior and access through interfaces.

Communication paradigms: deals with how entities communicate in a distributed system, and consider three communication paradigm:

● Inter-process communication

● Remote invocation

● Indirect communication.

Inter-process communication refers to the relatively low-level support for communication between processes in distributed systems, including message-passing primitives, direct access to the API offered by Internet protocols (socket programming) and support for multicast communication.

Remote invocation represents the most common communication paradigm in distributed systems, covering a range of techniques based on a two-way exchange between communicating entities in a distributed system and resulting in the calling of a remote operation (Request-*reply protocols).* Request-reply protocols are effectively a pattern imposed on an underlying procedure or method.

Request-reply protocols: Request-reply protocols are effectively a pattern imposed on an underlying message-passing service to support client-server computing. In particular, such protocols typically involve a pairwise exchange of messages from client to server and then from server back to client, with the first message containing an encoding of the operation to be executed at the server and also an array of bytes holding associated arguments and the second message containing any results of the operation,

-*Remote procedure calls* **:** In RPC, procedures in processes on remote computers can be called as if they are procedures in the local address space. The underlying RPC system then hides important aspects of distribution, including the encoding and decoding of parameters and results, the passing of messages and the preserving of the required semantics for the procedure call.

-*Remote method invocation:* Remote method invocation (RMI) strongly resembles remote procedure calls but in a world of distributed objects. With this approach, a calling object can invoke a method in a remote object. As with RPC, the underlying details are generally hidden from the user.

Indirect communication through a third entity, allowing a strong degree of decoupling between senders and receivers. In particular:

● Senders do not need to know who they are sending to (space uncoupling).

● Senders and receivers do not need to exist at the same time (time uncoupling).

Key techniques for indirect communication include:

Group communication: Group communication is concerned with the delivery of messages to a set of recipients and hence is a multiparty communication paradigm supporting one-to-many communication. Group communication relies on the abstraction of a group which is represented in the system by a group identifier.

Publish-subscribe systems: Many systems, such as the financial trading example can be classified as information-dissemination systems wherein a large number of producers (or publishers) distribute information items of interest (events) to a similarly large number of consumers (or subscribers). Publish-subscribe systems all share the crucial feature of providing an intermediary service that efficiently ensures information generated by producers is routed to consumers who desire this information.

Message queues: Whereas publish-subscribe systems offer a one-to-many style of communication; message queues offer a point-to-point service whereby producer processes can send messages to a specified queue and consumer processes can receive messages from the queue or be notified of the arrival of new messages in the queue.

Tuple spaces: Tuple spaces offer a further indirect communication service by supporting a model whereby processes can place arbitrary items of structured data, called tuples, in a persistent tuple space and other processes can either read or remove such tuples from the tuple space by specifying patterns of interest.

Distributed shared memory: Distributed shared memory (DSM) systems provide an abstraction for sharing data between processes that do not share physical memory. Programmers are nevertheless presented with a familiar abstraction of reading or writing (shared) data structures as if they were in their own local address spaces, thus presenting a high level of distribution transparency.

2.1.2 System Architectures: deals with the roles and responsibility

● Client- server model

● Services provided by multiple servers.

● Proxy servers and caches.

● Peer processes

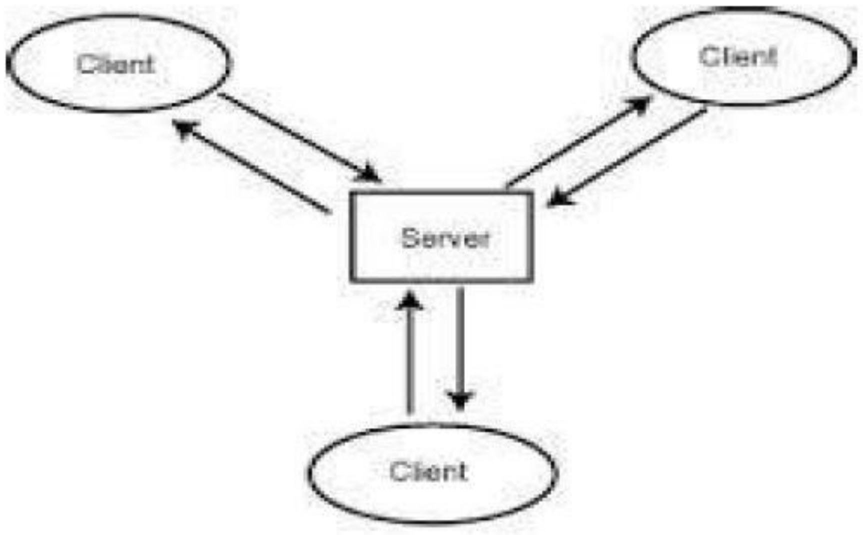
Client- server model

The system is structured as a set of processes, called servers, that offer services to the users, called clients. The client-server model is usually based on a simple request/reply protocol, implemented with send/receive.

● The client sends a request message to the server asking for some service.

● The server does the work and return a result or error code if the work could not be performed.

Web servers and most other Internet services are clients of the DNS service, which translates Internet domain names to network addresses. Another web-related example concerns search engines, which enable users to look up summaries of information available on web pages at sites throughout the Internet. These summaries are made by programs called web crawlers, which run in the background at a search engine site using HTTP requests to access web servers throughout the Internet.

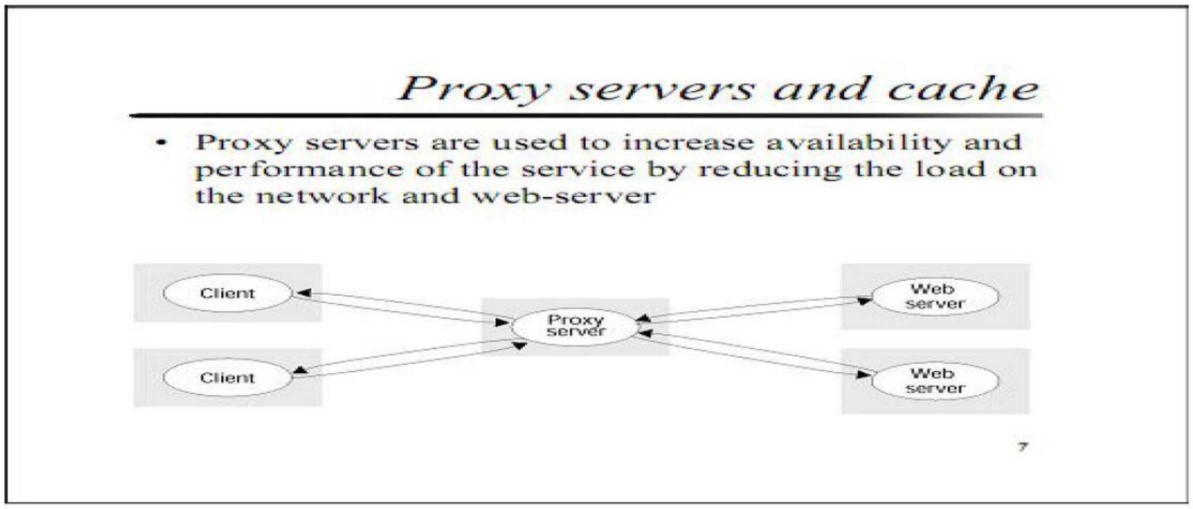


Services provided by multiple servers

Services may be implemented as several server processes in separate host computers interacting as necessary to provide a service to client processes. The Web provides a common example of partitioned data in which each web server manages its own set of resources. A user can employ a browser to access a resource at any one of the servers.

Proxy servers and caches

Proxy servers are used to increase availability and performance of the services by reducing the load on the network and web-server. Proxy server provides copies(replications) of resources which are managed by other server. Web browsers maintain a cache of recently visited web pages and other web resources in the client's local file system, using a special HTTP request to check with the original server that cached pages are up-to-date before displaying them.



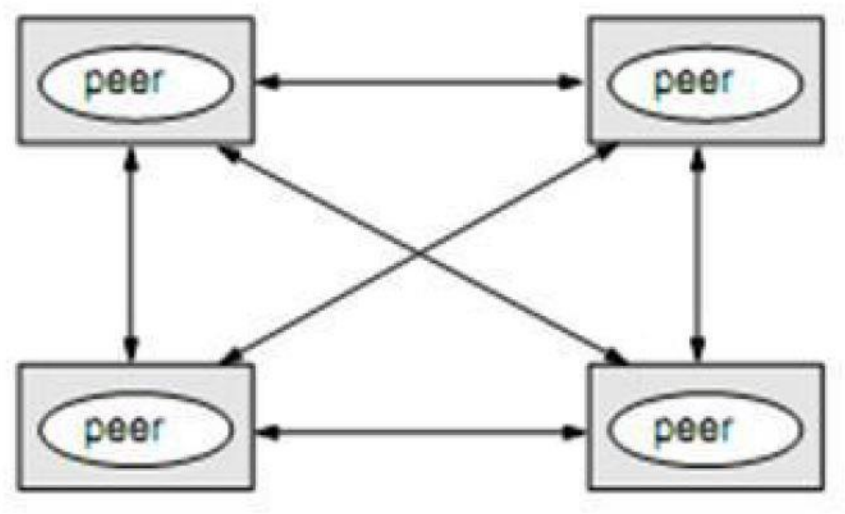
Proxy servers and cache

- Proxy servers are used to increase availability and performance of the service by reducing the load on the network and web-server

Client
Client
Proxy server
Web server
Web server
7

Peer processes

All processes (objects) play similar roles without distinction between client or servers. It distributes shared resources widely and it share computing and communication loads.



2.1.3 Variations on the client-server model (mapping)

● Mobile code

● Mobile agent

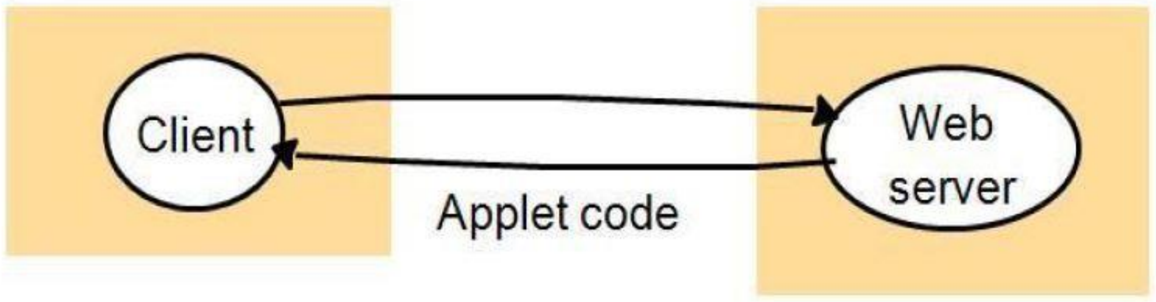
● Thin client

● Network computers

Mobile code: It is used to refer to code that can be sent from one computer to another and run at the destination. Its advantage is remote invocations are replaced by local ones so no need to suffer from the delays. Example: java applets

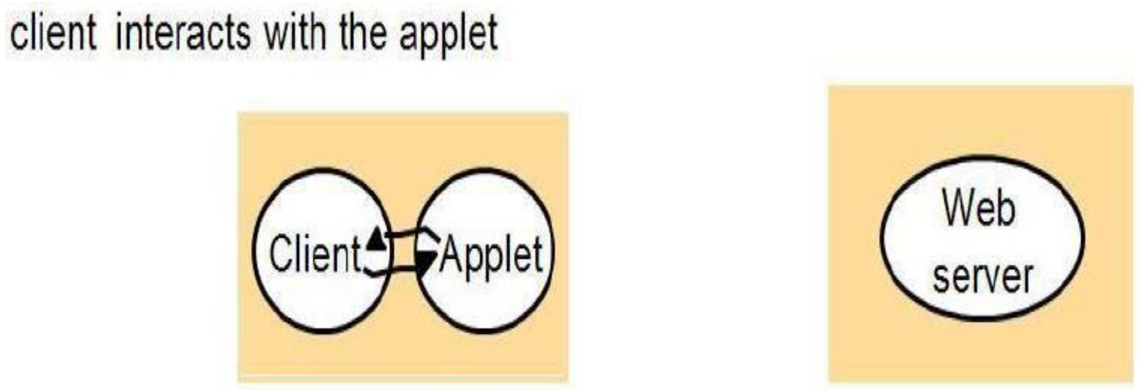
Step: 1

The user running a browser selects a link to applets whose code is stored on a web server. The code is downloaded to the browser and runs there.



Step: 2

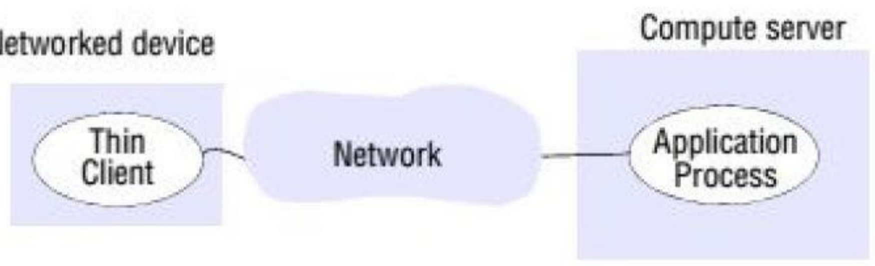
Client interacts with the applet.



Mobile agent: It is a running program that travels from one computer to another carrying out a task to someone’s behalf , such as collecting information, eventually returning with the results

Thin client: A thin client is a lightweight computer that establish a remote connection with a server-based computing environment. This architecture has low management and hardware cost. Here instead of downloading applications into user's computer, it runs on computer server.

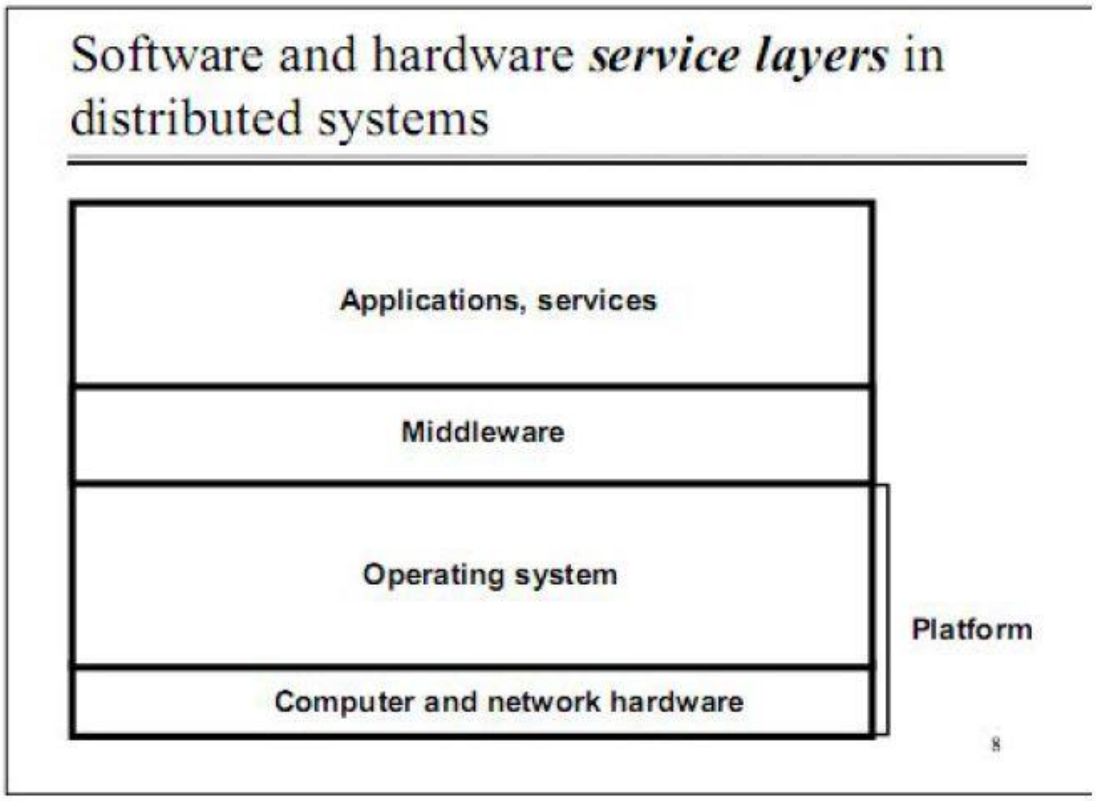
Thin clients and computer servers



Network computers: It downloads its operating system and any application software needed by the user from a remote server. Applications are run locally but files are managed by remote file server.

2.1.4 Software layers:

The layered architecture expressed in term of service layers offers a software abstraction, with higher layers being unaware of implementation details, or indeed of any other layers beneath them.


Software and hardware *service layers* in distributed systems
Applications, services
Middleware
Operating system
Computer and network hardware
Platform
8

Platform for distributed systems and applications consists of the lowest-level hardware and software layers. These low-level layers provide services to the layers above them, which are implemented independently in each computer, bringing the system's programming interface up to a level that facilitates communication and coordination between processes. Intel x86/Windows, Intel x86/Solaris, Intel x86/Mac OS X, Intel x86/Linux and ARM/Symbian are major examples.

Middleware was defined as a layer of software whose purpose is to mask heterogeneity and to provide a convenient programming model to application programmers. The activities of a middleware such as:

➢ Remote method invocation

➢ Communication between groups of processes

➢ Notification of events

➢ The partitioning, placement and retrieval of shared data objects amongst cooperating computers;

➢ The replication of shared data objects;

➢ The transmission of multimedia data in real time.

Tiered architecture - Tiering is a technique to organize functionality of a given layer and place this functionality into appropriate servers and, as a secondary consideration, on to physical nodes.

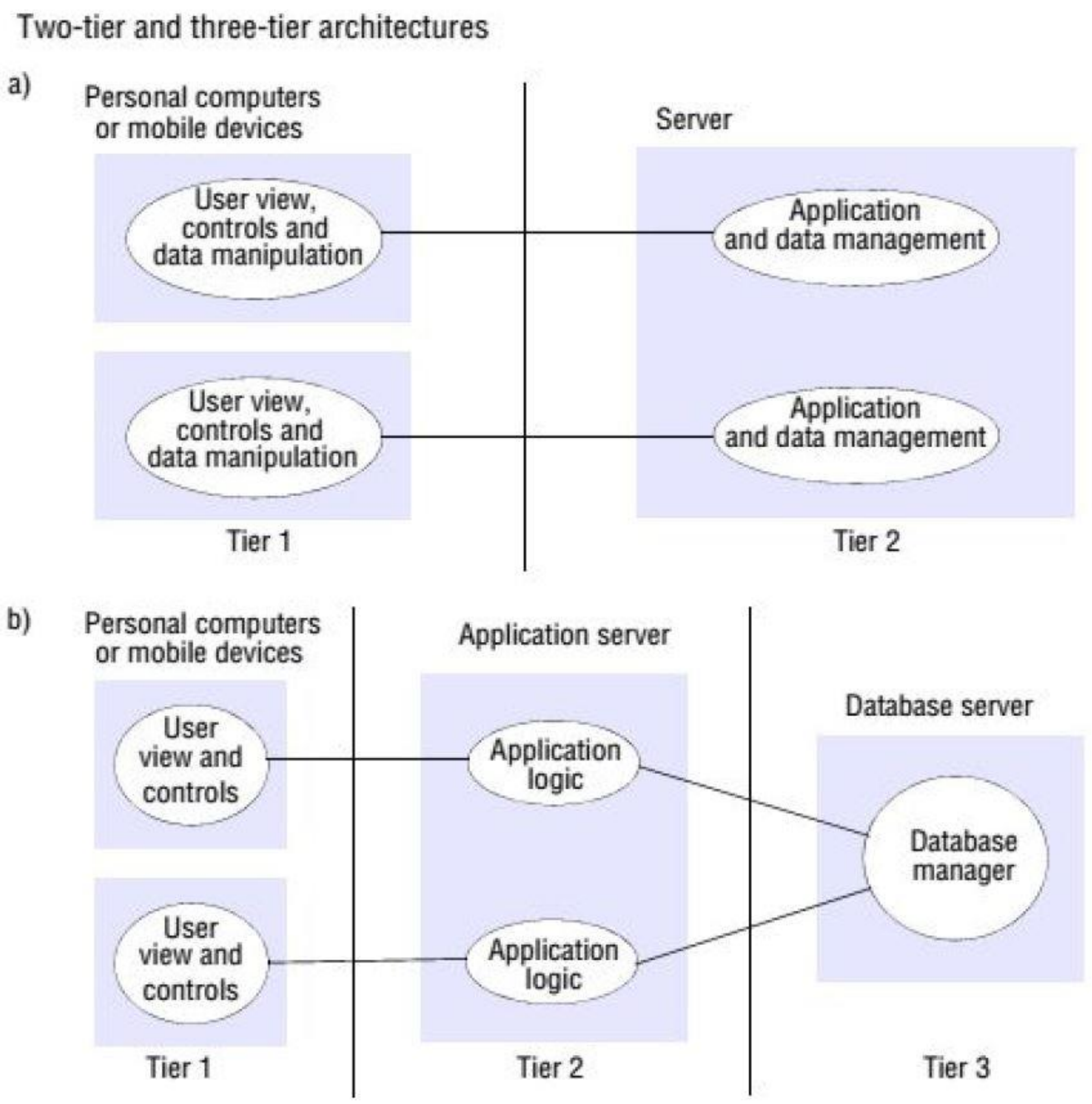
The concepts of two- and three-tiered architecture are:

● **The presentation logic,** which is concerned with handling user interaction and updating the view of the application as presented to the user.

● **The application logic,** which is concerned with the detailed application-specific processing associated with the application (also referred to as the business logic, although the concept is not limited only to business applications).

● **The data logic,** which is concerned with the persistent storage of the application, typically in a database management system.

Comparison between two-tier and three-tier architecture:


Two-tier and three-tier architectures
a)
Personal computers or mobile devices
Server
User view, controls and data manipulation
Application and data management
User view, controls and data manipulation
Application and data management
Tier 1
Tier 2
b)
Personal computers or mobile devices
Application server
Database server
User view and controls
Application logic
Database manager
User view and controls
Application logic
Tier 1
Tier 2
Tier 3

2.2 Fundamental Models:

Fundamental models deal with formal description of the properties that is common to architecture models. Specific about the characteristic and the failures and security risks they might exhibit.

● Interaction Models – processes interact by passing msgs and coordination b/w them.
● Failure Models – defines and classifies the failures.
● Security Models – modular nature and openness of the DS exposes it to the external and internal attacks.

2.2.1 Interaction Models:

Computation occurs within processes; the processes interact by passing messages, resulting in communication (information flow) and coordination (synchronization and ordering of activities) between processes. Interacting processes in a distributed system are affected by two significant factors:

1. Performance of communication channels

● **Latency:** delay between sending and receipt of a message
● **Jitter:** *jitter* is the deviation from true periodicity of a presumably periodic signa
● **Throughput:** No. of Packet send per unit time
● **Bandwidth:** total no. of information send per unit time

2. Computer clocks:

Each computer in a distributed system has its own internal clock to supply the value of the current time to local processes. Therefore, two processes running on different computers read their clocks at the same time may take different time values. Clock drift rate refers to the relative amount a computer clock differs from a perfect reference clock.

Two variants of the interactive model:

-Synchronous distributed systems

-Asynchronous distributed systems

Synchronous distributed systems:

● The time to execute each step of a process has known lower and upper bounds.

• Each message transmitted over a channel is received within a known bounded time.

• Each process has a local clock whose drift rate from real time has a known bound.

Asynchronous distributed systems: A system in which there are no bounds on:

● process execution times.

● message delivery times.

● clock drift rate.

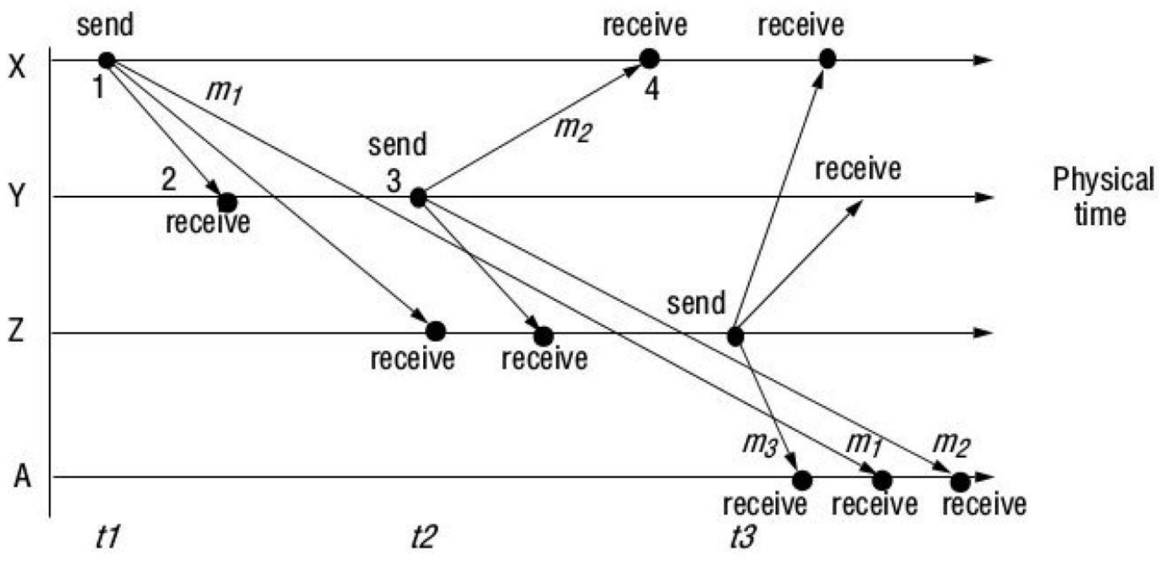
Event ordering:

Consider the following set of exchanges between groups of email

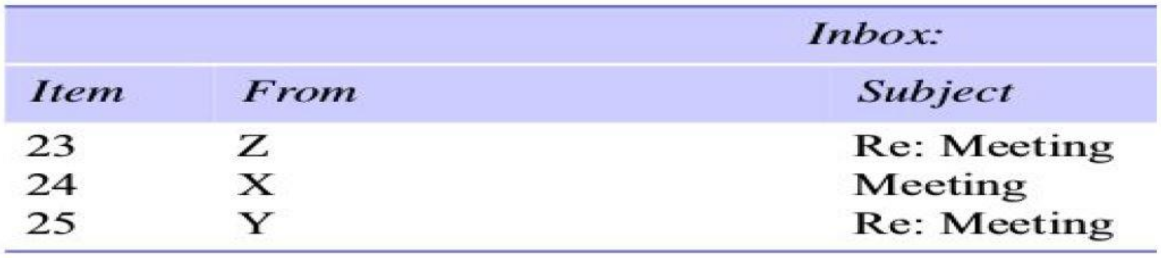
● users, X, Y, Z and A, on a mailing list:

1. User X sends a message with the subject Meeting.
2. Users Y and Z reply by sending a message with the subject Re: Meeting.

In real time, X's message is sent first, and Y reads it and replies; Z then reads both X's message and Y's reply and sends another reply, which references both X's and Y's messages. But due to the independent delays in message delivery, the messages may be delivered as shown in Figure.



If clocks cannot be synchronized perfectly across a distributed system, Lamport proposed a model of logical time that can be used to provide an ordering among the events at processes running in different computers in a distributed system. Logically, we know that a message is received after it was sent. Therefore we can state a logical ordering for pairs of events shown in Figure below:



		Inbox:
Item	From	Subject
23	Z	Re: Meeting
24	X	Meeting
25	Y	Re: Meeting

for example, considering only the events concerning X and Y:

X sends m 1 before Y receives m 1 ;

Y sends m 2 before X receives m 2 .

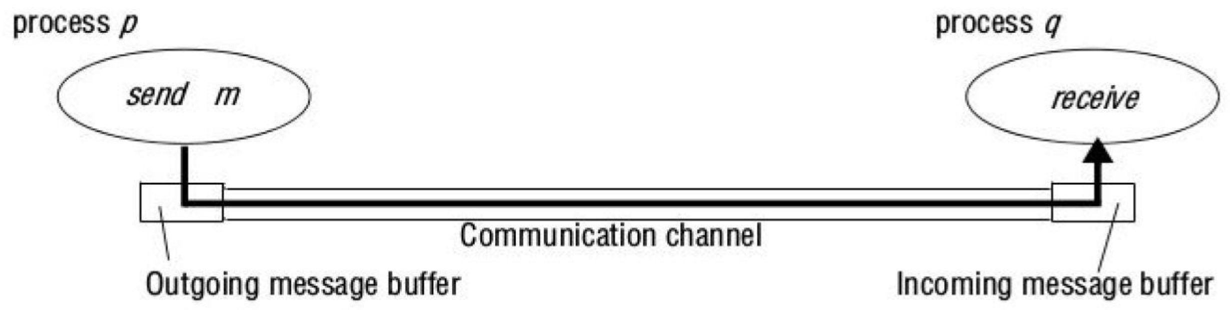
We also know that replies are sent after receiving messages, so we have the following logical ordering for Y:
Y receives m 1 before sending m 2 .

Logical time takes this idea further by assigning a number to each event corresponding to its logical ordering, so that later events have higher numbers than earlier ones. For example, Figure shows the numbers 1 to 4 on the events at X and Y.

2.2.2 Failure model

The failure model attempts to give a precise specification of the faults that can be exhibited by processes and communication channels. Failure may occur in order to provide an understanding of its effects, Including,

1. Omission Failures : Process or channel failed to do something. The chief omission failure of a process is crash and dropping message. When we say that a process has crashed we mean that it has halted and will not execute any further steps of its program ever. The communication channel produces an omission failure if it does not transport a message from p’s outgoing message buffer to q’s incoming message buffer. This is known as ‘dropping messages’



2. Arbitrary Failures/ Byzantine failure: Any Security Models type of error can occur in processes or channels. For example, a process may set wrong values in its data items, or it may return a wrong value in response to an invocation. An arbitrary failure of a process is one in which it arbitrarily omits intended processing steps or takes unintended processing steps

3. Timing Failures: Applicable only to synchronous distributed systems where time limits may not be met. Time limits are set to processes execution, communications and clock drifts rate. A timing faults occurs if any of this time limits exceeded.

4. Masking Failures: A service masks a failure by hiding it or converting it into a more acceptable type of failure. Checksums are used to mask corrupting messages -> an corrupted message is handled as a missing message. Message omission failures can be hidden by retransmitting messages.

The term reliable communication is defined in terms of validity and integrity as follows:

-Validity: Any message in the outgoing message buffer is eventually delivered to the incoming message buffer.

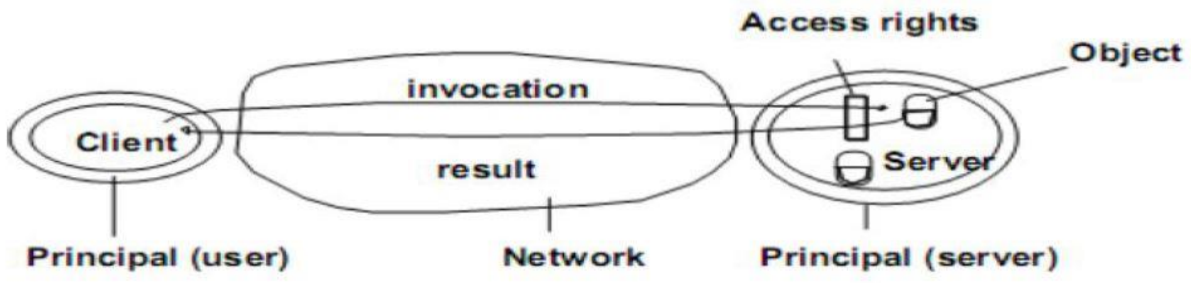
-Integrity: The message received is identical to one sent, and no messages are delivered twice.

2.2.3 Security Models

Security of a distributed system can be achieved by securing the processes and the channels used for their interactions and by protecting the objects that they encapsulate against unauthorized access. Security model includes,

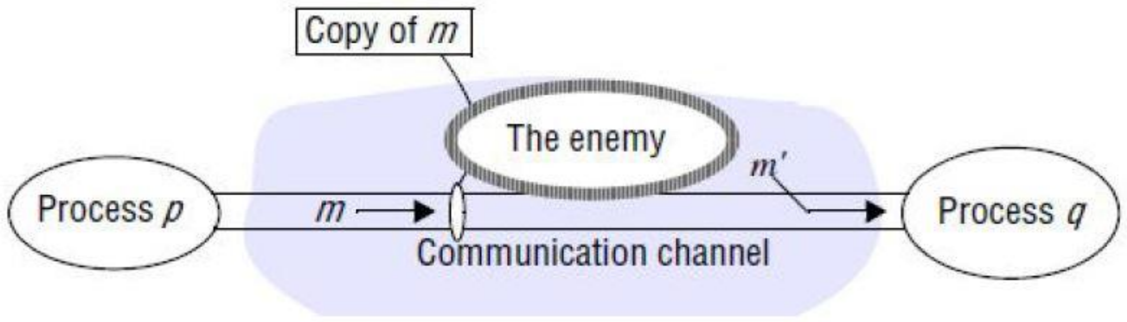
● **Protecting Objects**

1. Access Rights: who is allowed to perform the operations of an object.
2. Principal: the authority who has some rights on the object.



● **Securing processes and their interactions.**

1. The enemy: The threats from a potential enemy include threats to processes and threats to communication channels. Threats to processes: A process that is designed to handle incoming requests may receive a message from any other process in the distributed system, and it cannot necessarily determine the identity of the sender. Threats to communication channels: An enemy can copy, alter or inject messages as they travel across the network and its intervening gateways. Such attacks present a threat to the privacy and integrity of information as it travels over the network and to the integrity of the system.



2. Defeating security threats

● **Cryptography and shared secrets:** cryptography is the science of keeping messages secure, and encryption is the process of scrambling a message in such a way as to hide its contents.

● **Authentication:** The use of shared secrets and encryption provides the basis for the authentication of messages – proving the identities supplied by their senders.

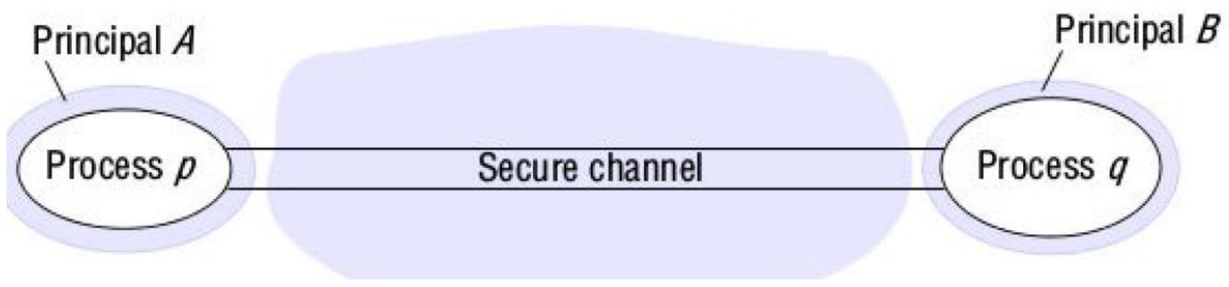
● **Secure channels:** Encryption and authentication are used to build secure channels as a service layer on top of existing communication services. A secure channel is a communication channel connecting a pair of processes, each of which acts on behalf of a principal.

A secure channel has the following properties:

● Each of the processes knows reliably the identity of the principal on whose behalf the other process is executing. Therefore if a client and server communicate via a secure channel

● A secure channel ensures the privacy and integrity (protection against tampering) of the data transmitted across it.

● Each message includes a physical or logical timestamp to prevent messages from being replayed or reordered



3. Other possible threats from an enemy

Two further security threats – denial of service attacks and the deployment of mobile code.

● ***Denial of service:*** This is a form of attack in which the enemy interferes with the activities of authorized users by making excessive and pointless invocations on services or message transmissions in a network, resulting in overloading of physical resources

● ***Mobile code:*** Mobile code raises new and interesting security problems for any process that receives and executes program code from elsewhere, such as the email attachment. Such code may easily play a Trojan horse role, purporting to fulfil an innocent purpose.

The uses of security models

The use of security techniques such as encryption and access control incurs substantial processing and management costs.

2.3 Physical model

Baseline of physical model is, a distributed system one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages. This leads to a minimal physical model of a distributed system as an extensible set of computer nodes interconnected by a computer network for the required passing of messages.

Three generations of distributed systems.

● Early distributed system

● Internet-scale distributed systems

● Contemporary distributed systems

2.3.1 Early distributed systems

Such systems emerged in the late 1970s and early 1980s in response to the emergence of local area networking technology. These systems typically consisted of between 10 and 100 nodes interconnected by a local area network, with limited Internet connectivity and supported a small range of services. Individual systems were largely homogeneous and openness was not a primary concern. Providing quality of service was still very much in its infancy and was a focal point for much of the research around such early systems.

2.3.2 Internet-scale distributed systems

Larger-scale distributed systems started to emerge in the 1990s in response to the dramatic growth of the Internet (for example, the Google search engine was first launched in 1996). In such systems, , an extensible set of nodes interconnected by a network of networks (the Internet). They incorporate large numbers of nodes and provide distributed system services for global organizations. The level of heterogeneity in such systems is significant in terms of networks, computer architecture, operating systems, languages employed and the development teams involved. This has led to an increasing emphasis on open standards

2.3.3 Contemporary distributed systems

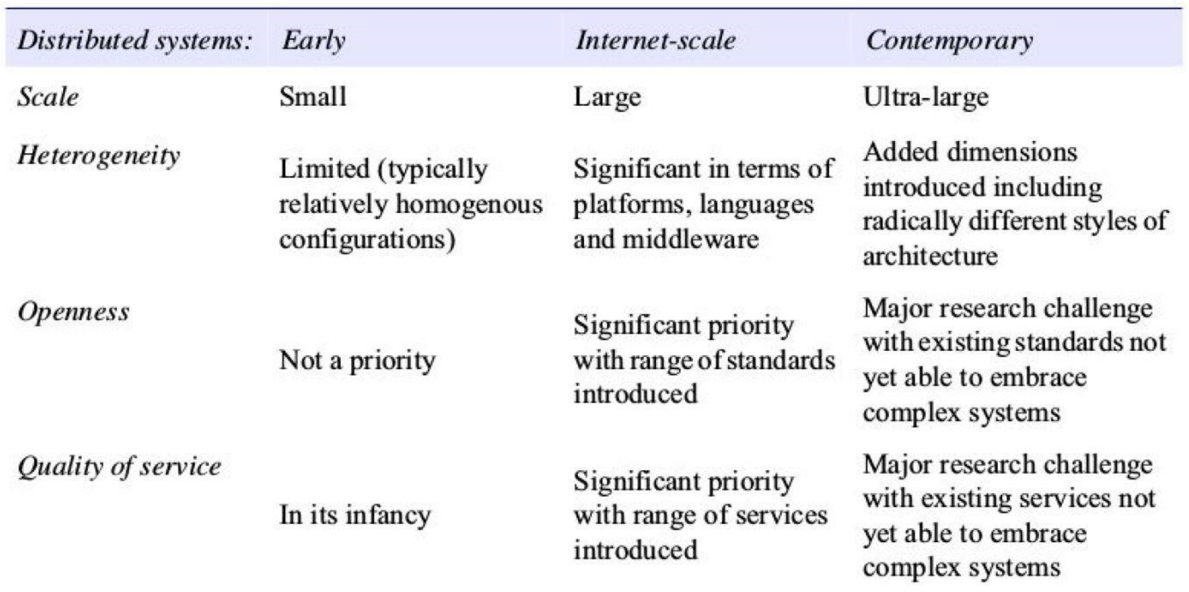
In the above systems, nodes were typically desktop computers and therefore relatively static, discrete (not embedded within other physical entities) and autonomous.

• The emergence of mobile computing has led to physical models where nodes such as laptops or smart phones may move from location to location in a distributed system

• The emergence of ubiquitous computing has led to a move from discrete nodes to architectures where computers are embedded in everyday objects and in the surrounding environment
• The emergence of cloud computing and, in particular, cluster architectures have led to a move from autonomous nodes performing a given role to pools of nodes that together provide a given service.

The end result is a physical architecture with a significant increase in the level of heterogeneity embracing, openness and quality of service. Such systems potentially involve up to hundreds of thousands of nodes.

Distributed systems of systems: The emergence of ultra- large-scale (ULS) distributed systems. A system of systems can be defined as a complex system consisting of a series of subsystems that are systems in their own right and that come together to perform a particular task



Distributed systems:	*Early*	*Internet-scale*	*Contemporary*
Scale	Small	Large	Ultra-large
Heterogeneity	Limited (typically relatively homogenous configurations)	Significant in terms of platforms, languages and middleware	Added dimensions introduced including radically different styles of architecture
Openness	Not a priority	Significant priority with range of standards introduced	Major research challenge with existing standards not yet able to embrace complex systems
Quality of service	In its infancy	Significant priority with range of services introduced	Major research challenge with existing services not yet able to embrace complex systems

MODULE 3

INTERPROCESS COMMUNICATION

Interprocess communication (IPC) is a set of programming interfaces that allow a programmer to coordinate activities among different program processes that can run concurrently in an operating system. It take place by message passing.

Application program interface (**API**) is a set of routines, protocols, and tools for building software applications. It includes,

1. The characteristics of Inter-process Communication.
2. Sockets
3. UDP Datagram Communication
4. TCP Stream Communication

The characteristics of Inter-process Communication.

Message passing between a pair of processes can be supported by two message communication operations, send and receive, defined in terms of destinations and messages. To communicate, one process sends a message (a sequence of bytes) to a destination and another process at the destination receives the message. This activity involves the communication of data from the sending process to the receiving process and may involve the synchronization of the two processes. Including,

1. Synchronous and Asynchronous Communication
2. Message Destinations
3. Reliability
4. Ordering

● Synchronous and Asynchronous Communication
A queue is associated with each message destination. Sending processes cause messages to be added to remote queues and receiving processes remove messages

from local queues. Communication between the sending and receiving processes may be either synchronous or asynchronous.

-In synchronous form of communication, the sending and receiving process synchronize at every message. In the synchronous form, both send and receive are blocking operations.

- In asynchronous form of communication, the sending operation is non-blocking and the receive operation can have blocking and non-blocking variants. The sending process is allowed to proceed as soon as the message has been copied to a local buffer, and the transmission of the message proceeds in parallel with the sending process. In the non-blocking variant, the receiving process proceeds with its program after issuing a receive operation, which provides a buffer to be filled in the background, but it must separately receive notification that its buffer has been filled, by polling or interrupt.

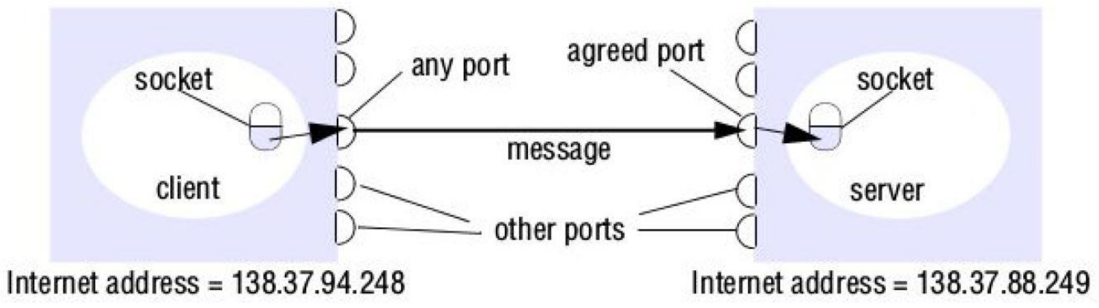
● **Message Destinations**
In the Internet protocols, messages are sent to local port. A local port is a message Destination within a computer, specified as an integer. A port has exactly one receiver (multicast ports are an exception) but can have many senders.

● **Reliability** : Reliable communication in defined terms of validity and integrity. A point to point message service is described as reliable- messages are guaranteed to be delivered despite a reasonable number of packets being dropped or lost. For integrity, messages must arrive uncorrupted and without duplication.

● **Ordering** : Some applications require that messages be delivered in sender order – that is, the order in which they were transmitted by the sender. The delivery of messages out of sender order is regarded as a failure by such applications.

Sockets

Processes can send and receive messages via a socket. Interprocess communication consists of transmitting a message between a socket in one process and a socket in another process. Sockets need to be bound to a port number and an Internet address in order to send and receive messages. Each socket has a transport protocol (TCP or UDP). Both form of communication, UDP and TCP, use the socket, which provides endpoint for communication between processes.


socket
any port
agreed port
socket
message
client
server
other ports
Internet address = 138.37.94.248
Internet address = 138.37.88.249

Java API for Internet addresses : As the IP packets underlying UDP and TCP are sent to Internet addresses, Java provides a class, **InetAddress** , that represents Internet addresses. Users of this class refer to computers by Domain Name System (DNS) host names.

InetAddress aComputer = InetAddress.getByName("bruno.dcs.qmul.ac.uk");

UDP Datagram Communication

The User Datagram Protocol (UDP) is a transport layer protocol. The service provided by UDP is an unreliable service that provides no guarantees for delivery and no protection from duplication. Some issues relating to datagram communication Including

- Message Size
- Blocking
- Timeouts

● Receive from any

Message Size: Receiving process specify an array of bytes to receive message. If size of the message is bigger than the array, then message is truncated.

Blocking: Sockets normally provide non-blocking sends and blocking receives for datagram communication. On arrival, the message is placed in a queue for the socket that is bound to the destination port. The message can be collected from the queue Messages are discarded at the destination if no process already has a socket bound to the destination port.

Timeouts: A process that has invoked a receive operation should wait indefinitely in situations where the sending process may have crashed or the expected message may have been lost. To allow for such requirements, timeouts can be set on sockets. Choosing an appropriate timeout interval is difficult, but it should be fairly large in comparison with the time required to transmit a message.

Receive from any: The receive method does not specify an origin for messages. Instead, an invocation of receive gets a message addressed to its socket from any origin.

Failure model for UDP datagrams : A failure model for communication channels and defines reliable communication in terms of two properties: integrity and validity. The integrity property requires that messages should not be corrupted or duplicated. The use of a checksum ensures that there is a negligible probability that any message received is corrupted. UDP datagrams suffer from the following failures:

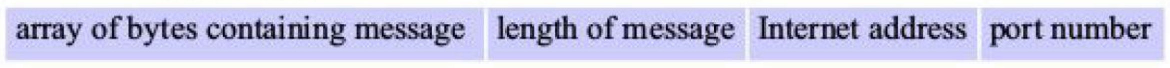
Omission failures: Messages may be dropped occasionally, either because of a checksum error or because no buffer space is available at the source or destination.

Use of UDP: the Domain Name System(DNS), Simple Network Management Protocol (**SNMP**) **,** Voice over IP **(VOIP)**

Java API for UDP datagrams: The Java API provides datagram communication by means of two classes: **DatagramPacket** and **DatagramSocket.**

DatagramPacket: This class provides a constructor that makes an instance out of an array
DatagramSocket: This class supports sockets for sending and receiving UDP datagrams

Datagram packet



array of bytes containing message	length of message	Internet address	port number

TCP Stream communication

TCP guarantees delivery of data and also guarantees that packets will be delivered in the same order in which they were sent. TCP is a connection-oriented protocol, which means a connection is established and maintained until the application programs at each end have finished exchanging messages.
The following characteristics of the network are hidden by the stream abstraction:

● **Message sizes:** The application can choose how much data it writes to a stream or reads from it. It may deal in very small or very large sets of data. The underlying implementation of a TCP stream decides how much data to collect before transmitting it as one or more IP packets.

● **Lost messages:** The TCP protocol uses an acknowledgement scheme. If the sender does not receive an acknowledgement within a timeout, it retransmits the message.

● **Flow control:** The TCP protocol attempts to match the speeds of the processes that read from and write to a stream. If the writer is too fast for the reader, then it is blocked until the reader has consumed sufficient data.

● **Message duplication and ordering:** Message identifiers are associated with each IP packet, which enables the recipient to detect and reject duplicates, or to reorder messages that do not arrive in sender order.

● **Message destinations:** A pair of communicating processes establishes a connection before they can communicate over a stream. Establishing a connection involves a connect request from client to server followed by an accept request from server to client before any communication can take place.

The API for stream communication assumes that when a pair of processes are establishing a connection, one of them plays the client role and the other plays the server role, but thereafter they could be peers. The client role involves creating a stream socket bound to any port and then making a connect request asking for a connection to a server at its server port. The server role involves creating a listening socket bound to a server port and waiting for clients to request connections. The listening socket maintains a queue of incoming connection requests. In the socket model, when the server accepts a connection, a new stream socket is created for the server to communicate with a client. The pair of sockets in the client and server are connected by a pair of streams, one in each direction. Thus each socket has an input stream and an output stream.

When an application closes a socket, this indicates that it will not write any more data to its output stream. Any data in the output buffer is sent to the other end of the stream and put in the queue at the destination socket, with an indication that the stream is broken. When a process exits or fails, all of its sockets are eventually closed and any process attempting to communicate with it will discover that its connection has been broken.

Failure model • To satisfy the integrity property of reliable communication, TCP streams use checksums to detect and reject corrupt packets and sequence numbers to detect and reject duplicate packets. For the sake of the validity property, TCP streams use timeouts and retransmissions to deal with lost packets. Therefore, messages are guaranteed to be delivered even when some of the underlying packets are lost. The TCP software responsible for sending messages will receive no acknowledgements and after a time will declare the connection to be broken.

Use of TCP:

● **HTTP** :-The Hypertext Transfer Protocol is used for communication between web browsers and web servers.
● **FTP:** The File Transfer Protocol allows directories on a remote computer to be browsed and files to be transferred from one computer to another over a connection.
● **Telnet:** Telnet provides access by means of a terminal session to a remote computer.
● **SMTP:** The Simple Mail Transfer Protocol is used to send mail between computers.

Java API for TCP streams: The Java interface to TCP streams is provided in the classes **ServerSocket** and **Socket** :

ServerSocket: This class is intended for use by a server to create a socket at a server port for listening for connect requests from clients.

Socket: This class is for use by a pair of processes with a connection.

3.2 Group communication

Group communication provides an example of an indirect communication paradigm. Group communication offers a service whereby a message is sent to a group and then this message is delivered to all members of the group. Group communication represents an abstraction over multicast communication With the added guarantees, group communication is to IP multicast what TCP is to
the point-to-point service in IP. Group communication is an important building block for distributed systems, and particularly reliable distributed systems, with key areas of application including:

● The reliable dissemination of information to potentially large numbers of clients, including in the financial industry, where institutions require accurate and up-to-date access to a wide variety of information sources.

● Support for collaborative applications, where again events must be disseminated to multiple users to preserve a common user view – for example, in multiuser games.

● Support for a range of fault-tolerance strategies, including the consistent update of replicated data or the implementation of highly available (replicated) servers.

● Support for system monitoring and management, including for example load balancing strategies.

The programming model

In group communication, the central concept is that of a group with associated *group membership* , whereby processes may join or leave the group. Processes can then send a message to this group and have it propagated to all members of the group with certain guarantees in terms of reliability and ordering. Thus, group communication implements *multicast communication,* in which a message is sent to all the members of the group by a single operation. Communication to all processes in the system, as opposed to a subgroup of them, is known as ***broadcast,*** whereas communication to a single process is known as ***unicast.***

Process groups and object groups• Most work on group services focuses on the concept of process groups, that is, groups where the communicating entities are processes. Such services are relatively low-level in that:

• Messages are delivered to processes and no further support for dispatching is provided.

• Messages are typically unstructured byte arrays with no support for marshalling of complex data types.

Object groups provide a higher-level approach to group computing. An object group is a collection of objects (normally instances of the same class) that process the same set of invocations concurrently, with each returning responses.

Types of groups

Closed and open groups *:* A group is said to be *closed* if only members of the group may multicast to it (Figure 3.2). A process in a closed group delivers to itself any message that it multicasts to the group. A group is *open* if processes outside the group may send to it.

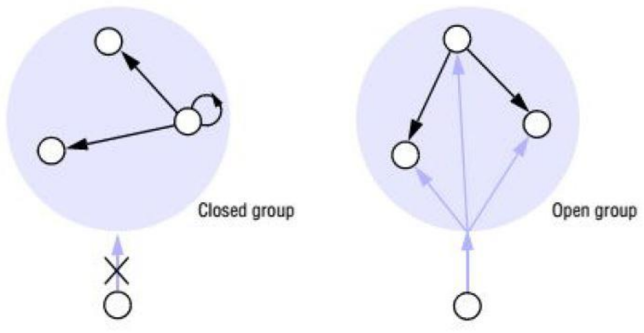


Figure 3.2 Open and closed groups

Closed groups of processes are useful, for example, for cooperating servers to send messages to one another that only they should receive. Open groups are useful, for example, for delivering events to groups of interested processes.

Overlapping and non-overlapping groups: In overlapping groups, entities (processes or objects) may be members of multiple groups, and non-overlapping groups imply that membership does not overlap.

Synchronous and asynchronous systems: There is a requirement to consider group communication in both environments.

Implementation issues

Implementation issues for group communication services, discussing the properties of the underlying multicast service in terms of reliability and ordering and also the key role of group membership management in dynamic environments.

1. Reliability and ordering in multicast • In group communication, all members of a group must receive copies of the messages sent to the group, generally with delivery guarantees. The guarantees include agreement on the set of messages that every process in the group should receive and on the delivery ordering across the group members. Reliability in one-to-one communication was defined in terms of two properties: *integrity* (the message received is the same as the one sent, and no messages are delivered twice) and *validity* (any outgoing message is eventually delivered).

The interpretation for reliable multicast builds on these properties, with integrity defined the same way in terms of delivering the message correctly at most once, and validity guaranteeing that a message sent will eventually be delivered. To extend the semantics to cover delivery to multiple receivers, a third property is added – that of agreement, stating that if the message is delivered to one process, then it is delivered to all processes in the group.

FIFO ordering : First-in-first-out (FIFO) ordering (also referred to as source ordering) is concerned with preserving the order from the perspective of a sender process, in that if a process sends one message before another, it will be delivered in this order at all processes in the group.

Causal ordering: Causal ordering takes into account causal relationships between messages, in that if a message happens before another message in the distributed system this so-called causal relationship will be preserved in the delivery of the associated messages at all processes.

***Total ordering* :** In total ordering, if a message is delivered before another message at one process, then the same order will be preserved at all processes.

2. Group membership management • The key elements of group communication management are summarized in Figure 3.3, which shows an open group.

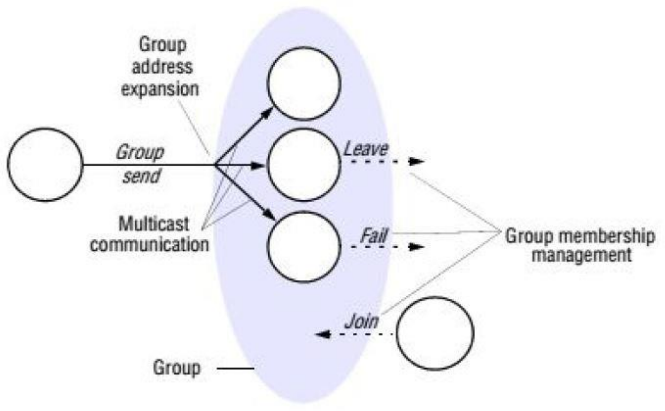


Figure 3.3 The role of group membership management

This diagram illustrates the important role of group membership management in maintaining an accurate view of the current membership, given that entities may join, leave or indeed fail. In more detail, a group membership service has four main tasks:

Providing an interface for group membership changes: The membership service provides operations to create and destroy process groups and to add or withdraw a process to or from a group. In most systems, a single process may belong to several groups at the same time (overlapping groups, as defined above). This is true of IP multicast, for example.

Failure detection **:** The service monitors the group members not only in case they should crash, but also in case they should become unreachable because of a communication failure. The detector marks processes as *Suspected or Unsuspected* . The service uses the failure detector to reach a decision about the group's membership: it excludes a process from membership if it is suspected to have failed or to have become unreachable.

Notifying members of group membership changes: The service notifies the group's members when a process is added, or when a process is excluded (through failure or when the process is deliberately withdrawn from the group).

Performing group address expansion: When a process multicasts a message, it supplies the group identifier rather than a list of processes in the group. The membership management service expands the identifier into the current group membership for delivery.

Case study: the JGroups toolkit

JGroups is a toolkit for reliable group communication written in Java. JGroups supports process groups in which processes are able to join or leave a group, send a message to all members of the group or indeed to a single member, and receive messages from the group.

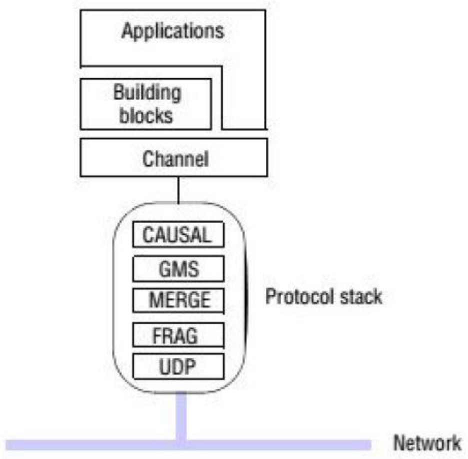


Figure 3.4 The architecture of JGroups

The architecture of JGroups is shown in Figure 3.4, which shows the main components of the JGroups implementation:

• Channels represent the most primitive interface for application developers, offering the core functions of joining, leaving, sending and receiving.

• Building blocks offer higher-level abstractions, building on the underlying service offered by channels. Examples of building blocks in JGroups are:

• *MessageDispatcher* is the most intuitive of the building blocks offered in JGroups. In group communication, it is often useful for a sender to send a message to a group and then wait for some or all of the replies. *MessageDispatcher* supports this by providing a *castMessage* method that sends a message to a group and blocks until a specified number of replies are received (for example, until a specified number n, a majority, or all messages are received).

• *RpcDispatcher* takes a specific method (together with optional parameters and results) and then invokes this method on all objects associated with a group. As with *MessageDispatcher* , the caller can block awaiting some or all of the replies.

• *NotificationBus* is an implementation of a distributed event bus, in which an event is any serializable Java object. This class is often used to implement consistency in replicated caches.

The protocol stack • JGroups follows the architectures offered by Horus and Ensemble by constructing protocol stacks out of protocol layers. In this approach, a protocol is a bidirectional stack of protocol layers with each layer implementing the following two methods:

public Object up (Event evt);
public Object down (Event evt);

Protocol processing therefore happens by passing events up and down the stack. In JGroups, events may be incoming or outgoing messages or management events, for example related to view changes. Each layer can carry out arbitrary processing on the message, including modifying its contents, adding a header or indeed dropping or reordering the message.
Protocol that consists of five layers:

• The layer referred to as UDP is the most common transport layer in JGroups. Note that, despite the name, this is not entirely equivalent to the UDP protocol; rather, the layer utilizes IP multicast for sending to all members in a group and UDP datagrams specifically for point-to-point communication. This layer therefore assumes that IP multicast is available.
If it is not, the layer can be configured to send a series of unicast messages to members, relying on another layer for membership discovery (in particular, a layer known as PING). For larger-scale systems operating over wide area networks, a TCP layer may be preferred (using the TCP protocol to send unicast messages and again relying on PING for membership discovery).

• FRAG implements message packetization and is configurable in terms of the maximum message size (8,192 bytes by default).

• MERGE is a protocol that deals with unexpected network partitioning and the subsequent merging of subgroups after the partition. A series of alternative merge layers are actually available, ranging from the simple to ones that deal with, for example, state transfer.

• GMS implements a group membership protocol to maintain consistent views of membership across the group

• CAUSAL implements causal ordering.

Multicast communication

Multicast operation is an operation that sends a single message from one process to each of the members of a group of processes. The simplest way of multicasting, provides no guarantees about message delivery or ordering. Multicasting has the following characteristics,

1. Fault tolerance based on replicated services
2. Finding the discovery servers in spontaneous networking.
3. Better performance through replicated data.
4. Propagation of event notifications

IP Multicast – an implementation of group communication.

IP multicast is built on top of the Internet protocol, IP. IP multicast allows the sender to transmit a single IP packet to a multicast group. A multicast group is specified by class D IP address for which first 4 bits are 1110 in Ipv4. The membership of a multicast group is dynamic, allowing computers to join or leave at any time and to join an arbitrary number of groups. It is possible to send datagrams to a multicast group without being a member. An application program performs multicasts by sending UDP datagrams with multicast addresses and ordinary port numbers. The following details are specific to IPv4:

● **Multicast routers:** IP packets can be multicast both on a local network and on the wider Internet. To limit the distance of propagation of a multicast datagram, the sender can specify the number of routers it is allowed to pass – called the time to live, or TTL for short.

● **Multicast address allocation:** Class D addresses (that is, addresses in the range 224.0.0.0 to 239.255.255.255) are reserved for multicast traffic and managed globally by the Internet Assigned Numbers Authority (IANA). This document defines a partitioning of this address space into a number of blocks, including:

Multicast addresses may be permanent or temporary. Permanent groups exist even when there are no members – their addresses are assigned by IANA and span the various blocks mentioned above. Addresses are reserved for a variety of purposes, from specific Internet protocols to given organizations that make heavy use of multicast traffic, including multimedia broadcasters and financial institutions. The remainder of the multicast addresses are available for use by temporary groups, which must be created before use and cease to exist when all the members have left.

Failure model for multicast datagrams • Datagrams multicast over IP multicast have the same failure characteristics as UDP datagrams – that is, they suffer from omission failures. The effect on a multicast is that messages are not guaranteed to be delivered to any particular group member in the face of even a single omission failure.

Java API to IP multicast • The Java API provides a datagram interface to IP multicast through the class ***MulticastSocket,*** which is a subclass of DatagramSocket with the additional capability of being able to join multicast groups. A process can join a multicast group with a given multicast address by invoking the ***joinGroup*** method of its multicast socket. A process can leave a specified group by invoking the
leaveGroup method of its multicast socket. The Java API allows the TTL to be set for a multicast socket by means of the ***setTimeToLive*** method.

Reliability and ordering of multicast

A datagram sent from one multicast router to another may be lost, thus preventing all recipients beyond that router from receiving the message. Also, when a multicast on a local area network uses the multicasting capabilities of the network to allow a single datagram to arrive at multiple recipients, any one of those recipients may drop the message because its buffer is full. Another factor is that any process may fail. If a multicast router fails, the group members beyond that router will not receive the multicast message, although local members may do so.
Ordering is another issue. IP packets sent over an internetwork do not necessarily arrive in the order in which they were sent, with the possible effect that some group members receive datagrams from a single sender in a different order from other group members. Characteristics are,

1. Fault tolerance based on replicated services: Consider a replicated service that consists of the members of a group of servers that start in the same initial state and always perform the same operations in the same order, so as to remain consistent with one another.

2.Discovering services in spontaneous networking : One way for a process to discover services in spontaneous networking is to multicast requests at periodic intervals, and for the available services to listen for those multicasts and respond.

3. Better performance through replicated data: Consider the case where the replicated data itself, rather than operations on the data, are distributed by means of multicast messages. The effect of lost messages and inconsistent ordering would depend on the method of replication and the importance of all replicas being totally up-to-date.

4. Propagation of event notifications: The particular application determines the qualities required of multicast.

Remote Procedure call

Applications composed of cooperating programs running in several different processes. Such programs need to invoke operations in other processes.

RPC – client programs call procedures in server programs, running in separate and remote computers

RMI – an object in one process can invoke methods of objects in another process

The earliest and perhaps the best-known programming model for distributed programming allows client programs to call procedures in server programs running in separate processes and generally in different computers from the client. RPC is a kind of request–response protocol. An RPC is initiated by the *client* , which sends a request message to a known remote *server* to execute a specified procedure with supplied parameters. The remote server sends a response to the client, and the application continues its process. While the server is processing the call, the client is blocked (it waits until the server has finished processing before resuming execution), unless the client sends an asynchronous request to the server.

Design issues for RPC

There are three issues that are important in understanding this concept:

• the style of programming promoted by RPC – programming with interfaces;
• the call semantics associated with RPC;
• the key issue of transparency and how it relates to remote procedure calls.

Programming with interfaces : Most modern programming languages provide a means of organizing a program as a set of modules that can communicate with one another. Communication between modules can be by means of procedure calls between modules or by direct access to the variables in another module. In order to control the possible interactions between modules, an explicit interface is defined for each module. The interface of a module specifies the procedures and the variables that can be accessed from other modules. Interfaces in distributed systems: In a distributed program, the modules can run in separate processes. In the client-server model, in particular, each server provides a set of procedures that are available for use by clients.

An RPC mechanism can be integrated with a particular programming language if it includes an adequate notation for defining interfaces. Interface definition languages (IDLs) are designed to allow procedures implemented in different languages to invoke one another.

RPC call semantics: The main choices are,

● Retry request message: Controls whether to retransmit the request message until either a reply is received or the server is assumed to have failed.

● Duplicate filtering: Controls when retransmissions are used and whether to filter out duplicate requests at the server.

● Retransmission of results: Controls whether to keep a history of result messages to enable lost results to be retransmitted without re-executing the operations at the server.

The choices of RPC invocation semantics are defined as follows,

● Maybe semantics: With maybe semantics, the remote procedure call may be Executed once or not at all.

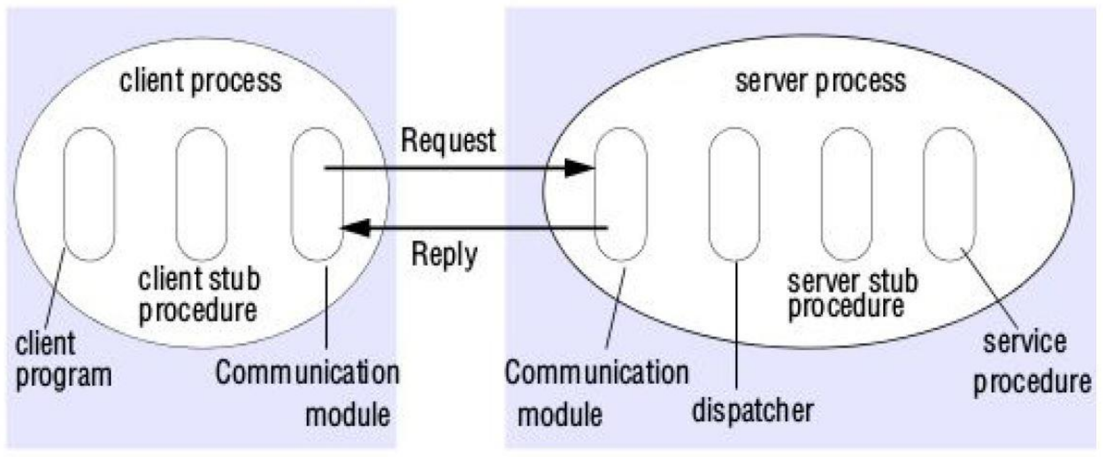
● At-least-once semantics: With at-least-once semantics, the invoker receives either a result, in which case the invoker knows that the procedure was executed at least once, or an exception informing it that no result was received. At-least-once semantics can be achieved by the retransmission of request messages.

● At-most-once semantics: With at-most-once semantics, the caller receives either a result, in which case the caller knows that the procedure was executed exactly once, or an exception informing it that no result was received, in which case the procedure will have been executed either once or not at all.

Implementation of RPC

The client that accesses a service includes one stub procedure for each procedure in the service interface. The stub procedure behaves like a local procedure to the client, but instead of executing the call, it marshals the procedure identifier and the arguments into a request message, which it sends via its communication module to the server. When the reply message arrives, it un-marshals the results. The server process contains a dispatcher together with one server stub procedure and one service procedure for each procedure in the service interface. The dispatcher selects one of the server stub procedures according to the procedure identifier in the request message. The server stub procedure then unmarshals the arguments in the request message, calls the corresponding service procedure and marshals the return values for the reply message.

Role of client and server stub procedures in RPC



Sequence of events

1. The client calls the client stub. The call is a local procedure call, with parameters pushed on to the stack in the normal way.

2. The client stub packs the parameters into a message and makes a system call to send the message. Packing the parameters is called marshalling .

3. The client's communication module sends the message from the client machine to the server machine.

4. The communication module on the server machine passes the incoming packets to the dispatcher.

5. The dispatcher selects one of the server stub procedures according to the procedure identifier in the request message.

6. The server stub unpacks the parameters from the message. Unpacking the parameters is called unmarshalling .

7. Finally, the server stub calls the server procedure. The reply traces the same steps in the reverse direction.

Network virtualization

Network virtualization is concerned with the construction of many different virtual networks over an existing network such as the Internet. Each virtual network can be designed to support a particular distributed application. It would be impractical to attempt to alter the Internet protocols to suit each of the many applications running over them – what might enhance one of them could be detrimental to another. In addition, the IP transport service is implemented over a large and ever- increasing number of network technologies. These two factors have led to the interest in network virtualization.

Overlay networks

An overlay network is a virtual network consisting of nodes and virtual links, which sits on top of an underlying network (such as an IP network) and offers something that is not otherwise provided:

A service that is tailored towards the needs of a class of application or a particular higher-level service – for example, multimedia content distribution.

More efficient operation in a given networked environment – for example routing in an ad hoc network

An additional feature – for example, multicast or secure communication.
Overlay networks have the following advantages:

• They enable new network services to be defined without requiring changes to the underlying network, a crucial point given the level of standardization in this area and the difficulties of amending underlying router functionality.

• They encourage experimentation with network services and the customization of services to particular classes of application.

• Multiple overlays can be defined and can coexist, with the end result being a more open and extensible network architecture.

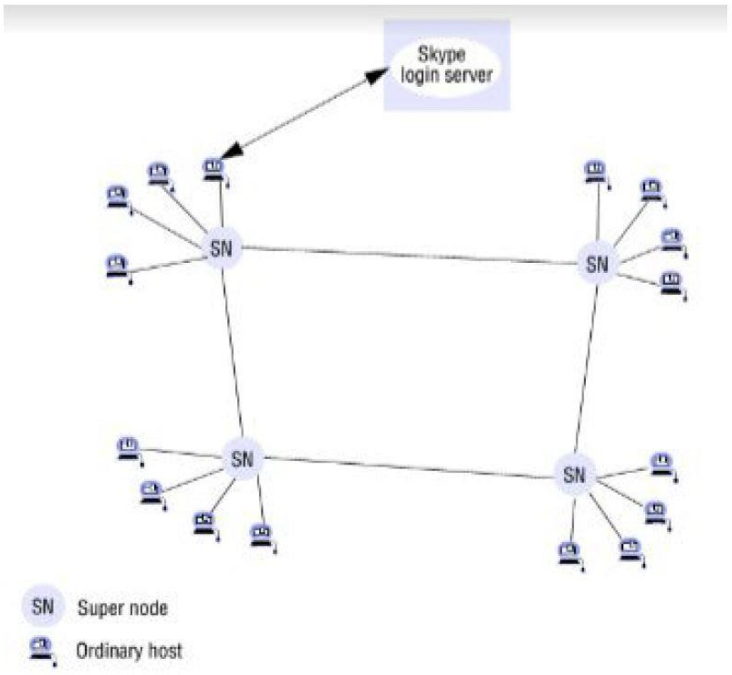
Skype: An example of an overlay network

Skype is a peer-to-peer application offering Voice over IP (VoIP). It also includes instant messaging, video conferencing and interfaces to the standard telephony service through SkypeIn and SkypeOut. Skype is an excellent case study of the use of overlay networks in real world systems, indicating how advanced functionality can be provided in an applicationspecific manner and without modification of the core architecture of the Internet.

Skype is a virtual network in that it establishes connections between people. No IP address or port is required to establish a call. The architecture of the virtual network supporting Skype is not widely publicized but researchers have studied Skype through a variety of methods, including traffic analysis, and its principles are now in the public domain.

Skype architecture

Skype is based on a peer-to-peer infrastructure consisting of ordinary users' machines (referred to as hosts) and super nodes – super nodes are ordinary Skype hosts that happen to have sufficient capabilities to carry out their enhanced role. Super nodes are selected on demand based a range of criteria including bandwidth available, reachability (the machine must have a global IP address and not be hidden behind a NAT- enabled router, for example) and availability (based on the length of time that Skype has been running continuously on that node).



User connection • Skype users are authenticated via a well-known login server. They then make contact with a selected super node. To achieve this, each client maintains a cache of super node identities (that is, IP address and port number pairs). At first login this cache is filled with the addresses of around seven super nodes, and over time the client builds and maintains a much larger set (perhaps several hundred).

Search for users • The main goal of super nodes is to perform the efficient search of the global index of users, which is distributed across the super nodes. The search is orchestrated by the client's chosen super node and involves an expanding search of other super nodes until the specified user is found. On average, eight super nodes are contacted. A user search typically takes between three and four seconds to complete for hosts that have a global IP address (and slightly longer, five to six seconds, if behind a NAT-enabled router).

Voice connection • Once the required user is discovered, Skype establishes a voice connection between the two parties using TCP for signalling call requests and terminations and either UDP or TCP for the streaming audio. UDP is preferred but TCP, along with the use of an intermediary node, is used in certain circumstances to circumvent firewalls. The software used for encoding and decoding audio plays a key part in providing the excellent call quality normally attained using Skype, and the associated algorithms are carefully tailored to operate in Internet environments at 32 kbps and above.

MODULE 4

Distributed file Systems: Introduction

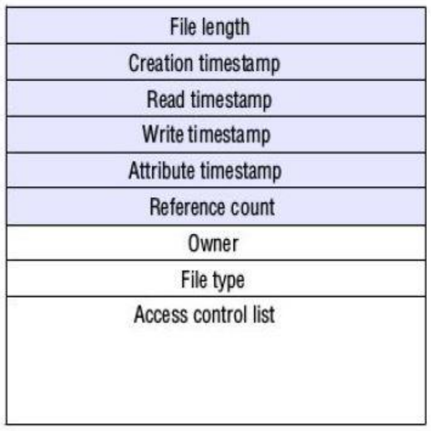
- A file system is a subsystem of the operating system that performs file management activities such as organization, storing, retrieval, naming, sharing, and protection of files.

- A distributed file system (DFS) is a method of storing and accessing files based in a client/server architecture. In a distributed file system, one or more central servers store files that can be accessed, with proper authorization rights, by any number of remote clients in the network.

- Files contain both data and attributes. The data consist of a sequence of data items, accessible by operations to read and write any portion of the sequence.

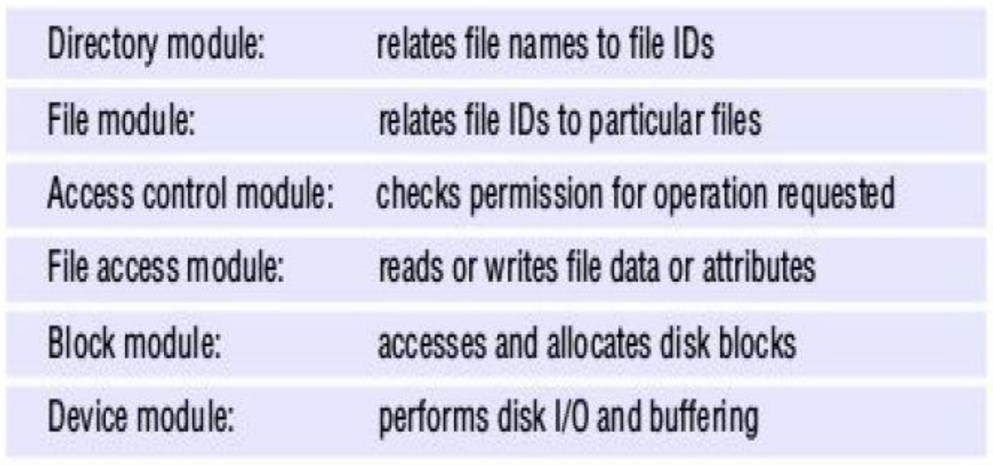
- The attributes are held as a single record containing information such as the length of the file, timestamps, file type, owner's identity and access control lists. The shaded attributes in the Fig. are managed by the file system and are not normally update by user programs.

File attribute record structure



- The term metadata is often used to refer to all of the extra information stored by a file system that is needed for the management of files. It includes file attributes, directories and all the other persistent information used by the file system.

File system modules



Directory module:	relates file names to file IDs
File module:	relates file IDs to particular files
Access control module:	checks permission for operation requested
File access module:	reads or writes file data or attributes
Block module:	accesses and allocates disk blocks
Device module:	performs disk I/O and buffering

Distributed file system requirements

(a) **Transparency:** as the concealment from the user and the application programmer of the separation of component in a DS.
- Access Transparency: Client programs should be unaware of the distribution of files. A single set of operations is provided for access to local and remote files.
- Location transparency: enable files to be accessed without knowledge of location
- Mobility transparency: allow the movement of file without affecting the operation of user
- Performance T: Client programs should continue to perform satisfactorily while the load on the service varies within a specified range.
- Scaling transparency: The service can be expanded by incremental growth to deal with a wide range of loads and network sizes.

(b) Concurrent file updates : Changes to a file by one client should not interfere with the operation of other clients simultaneously accessing or changing the same file.
(c) File replication : In a file service that supports replication, a file may be represented by several copies of its contents at different locations. It enhances fault tolerance by enabling clients to locate another server that holds a copy of the file when one has failed.

(d) Heterogeneity : The service interfaces should be defined so that client and server software can be implemented for different operating systems and computers.

(e) Fault Tolerance: Service continue to operate in face of failure.

(f) Security: In distributed file systems, there is a need to authenticate client requests so that access control at the server is based on correct user identities and to protect the contents of request and reply messages with digital signatures and (optionally) encryption of secret data.

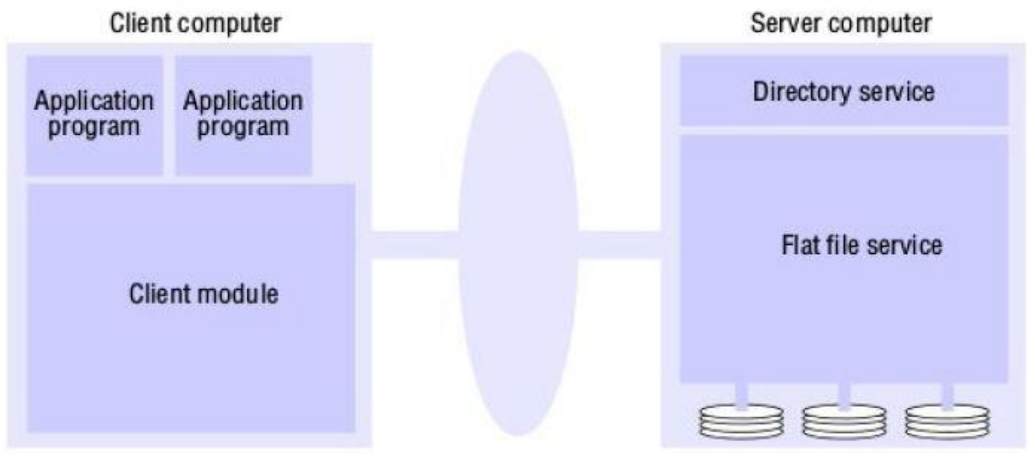
(g) Efficiency: Provide good level of performance

(h) Consistency: If any changes made to one file, that changes must do in other replicated copies.

File service architecture

An architecture that offers a clear separation of the main concerns in providing access to files is obtained by structuring the file service as three components – a flat file service, a directory service and a client module.

File service architecture



Flat file service • The flat file service is concerned with implementing operations on the contents of files. Unique file identifiers (UFIDs) are used to refer to files in all requests for flat file service operations. When the flat file service receives a request to create a file, it generates a new UFID for it and returns the UFID to the requester. In comparison with the UNIX interface, our flat file service has no open and close operations – files can be accessed immediately by quoting the appropriate UFID. The interface to our flat file service differs from the UNIX file system interface mainly for reasons of fault tolerance:

Repeatable operations: With the exception of Create, the operations are idempotent, Repeated execution of Create produces a different new file for each call.

Stateless servers: The interface is suitable for implementation by stateless servers. Stateless servers can be restarted after a failure and resume operation without any need for clients or the server to restore any state.

Figure 12.6 Flat file service operations

Read(FileId, i, n) → *Data* — throws *BadPosition*	If *1* ≤ *i* ≤ *Length(File)*: Reads a sequence of up to *n* items from a file starting at item *i* and returns it in *Data*.
Write(FileId, i, Data) — throws *BadPosition*	If *1* ≤ *i* ≤ *Length(File)+1*: Writes a sequence of *Data* to a file, starting at item *i*, extending the file if necessary.
Create() → *FileId*	Creates a new file of length 0 and delivers a UFID for it.
Delete(FileId)	Removes the file from the file store.
GetAttributes(FileId) → *Attr*	Returns the file attributes for the file.
SetAttributes(FileId, Attr)	Sets the file attributes (only those attributes that are not shaded in Figure 12.3).

Directory service: The directory service provides a mapping between text names for files and their UFIDs. Clients may obtain the UFID of a file by quoting its text name to the directory service.

Figure 12.7 Directory service operations

Lookup(Dir, Name) → *FileId* — throws *NotFound*	Locates the text name in the directory and returns the relevant UFID. If *Name* is not in the directory, throws an exception.
AddName(Dir, Name, FileId) — throws *NameDuplicate*	If *Name* is not in the directory, adds *(Name, File)* to the directory and updates the file's attribute record. If *Name* is already in the directory, throws an exception.
UnName(Dir, Name) — throws *NotFound*	If *Name* is in the directory, removes the entry containing *Name* from the directory. If *Name* is not in the directory, throws an exception.
GetNames(Dir, Pattern) → *NameSeq*	Returns all the text names in the directory that match the regular expression *Pattern*.

Client module • A client module runs in each client computer, integrating and extending the operations of the flat file service and the directory service under a single application programming interface that is available to user-level programs in client computers.

Access control:

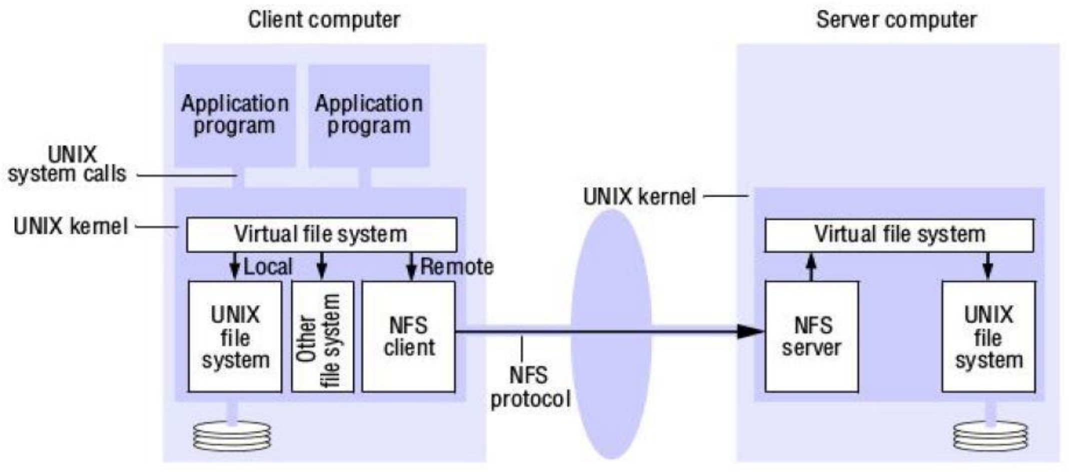
● An access check is made whenever a file name is converted to a UFID

● A user identity is submitted with every client request, and access checks are performed by the server for every file operation.

Hierarchic file system • A hierarchic file system such as the one that UNIX provides consists of a number of directories arranged in a tree structure. Each directory holds the names of the files and other directories that are accessible from it.

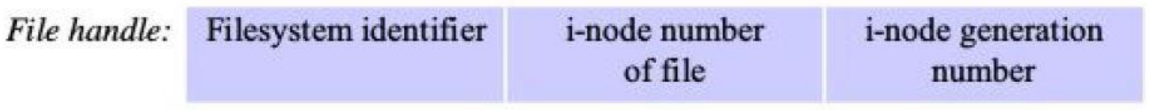
File groups • A file group is a collection of files located on a given server. A server may hold several file groups, and groups can be moved between servers, but a file cannot change the group to which it belongs.

Sun Network File System: Sun NFS



The NFS module resides in the kernel on each computer. Requests referring to files in a remote file system are translated by the client module to NFS protocol operations and then passed to the NFS server module at the computer holding the relevant file system. The NFS client and server modules communicate using remote procedure calls. The RPC interface to the NFS server is open: any process can send requests to an NFS server; if the requests are valid and they include valid user credentials, they will be acted upon.

Virtual file system: The integration is achieved by a virtual file system (VFS) module, which has been added to the UNIX kernel to distinguish between local and remote files. The file identifiers used in NFS are called file handles.



File handle:	Filesystem identifier	i-node number of file	i-node generation number

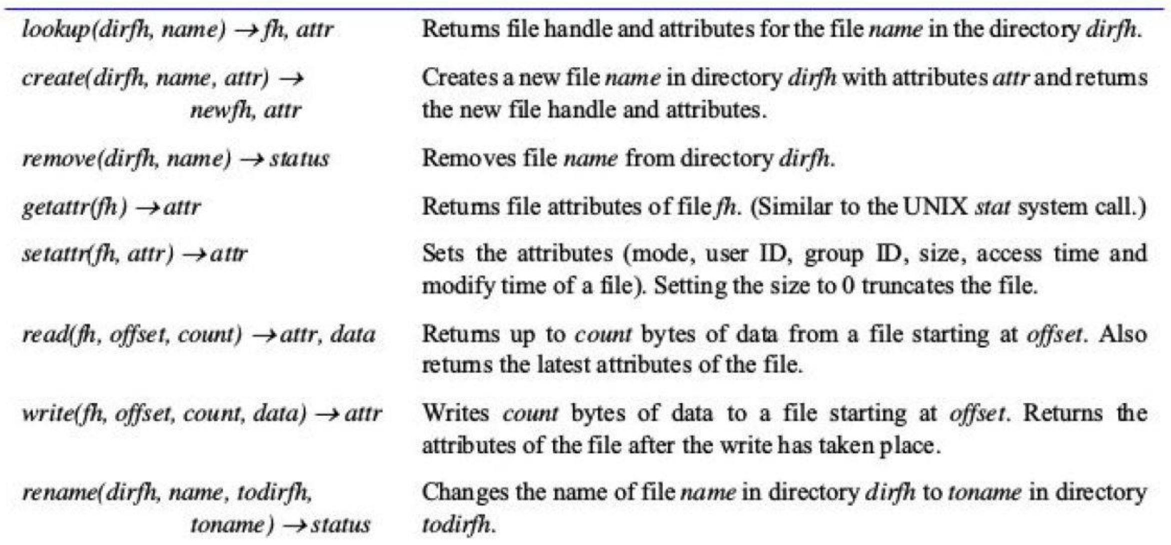
- The file system identifier field is a unique number that is allocated to each file system when it is created.
- The i-node number is needed to locate the file in file system and also used to store its attribute and i-node numbers are reused after a file is removed.
- The i-node generation number is needed to increment each time i-node numbers are reused after a file is removed. The virtual file system layer has one VFS structure for each mounted file system and one v-node per open file. The v-node contains an indicator to show whether a file is local or remote.

Client Integration: The NFS client module cooperates with the virtual file system in each client machine. It operates in a similar manner to the conventional UNIX file system, transferring blocks of files to and from the server and caching the blocks in the local memory whenever possible. If the file is local, the v-node contains a reference to the index of the local file . If the file is remote, it contains the file handle of the remote file.

Access control and authentication : the NFS server is stateless and does not keep files open on behalf of its clients. So the server must check the user's identity against the file's access permission attributes on each request, to see whether the user is permitted to access the file in the manner requested.

NFS server interface: The file and directory operations are integrated in a single service; the creation and insertion of file names in directories is performed by a single create operation, which takes the text name of the new file and the file handle for the target directory as arguments. The other NFS operations on directories are,

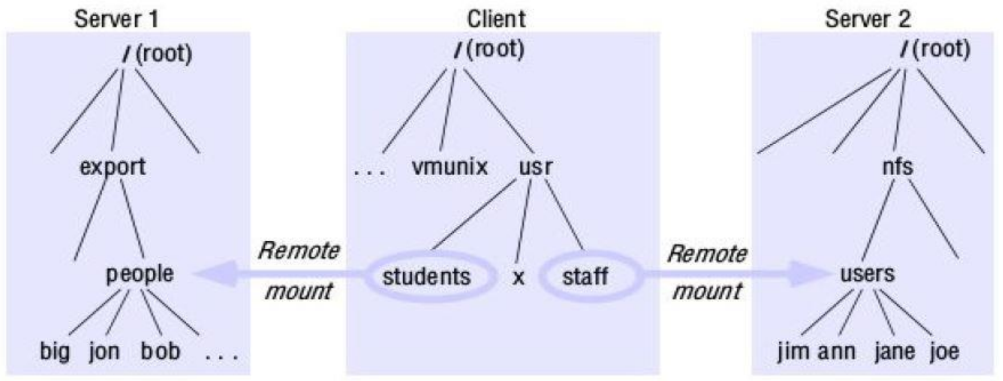
Figure 12.9 NFS server operations (NFS version 3 protocol, simplified)



lookup(dirfh, name) → fh, attr	Returns file handle and attributes for the file *name* in the directory *dirfh*.
create(dirfh, name, attr) → newfh, attr	Creates a new file *name* in directory *dirfh* with attributes *attr* and returns the new file handle and attributes.
remove(dirfh, name) → status	Removes file *name* from directory *dirfh*.
getattr(fh) → attr	Returns file attributes of file *fh*. (Similar to the UNIX *stat* system call.)
setattr(fh, attr) → attr	Sets the attributes (mode, user ID, group ID, size, access time and modify time of a file). Setting the size to 0 truncates the file.
read(fh, offset, count) → attr, data	Returns up to *count* bytes of data from a file starting at *offset*. Also returns the latest attributes of the file.
write(fh, offset, count, data) → attr	Writes *count* bytes of data to a file starting at *offset*. Returns the attributes of the file after the write has taken place.
rename(dirfh, name, todirfh, toname) → status	Changes the name of file *name* in directory *dirfh* to *toname* in directory *todirfh*.

Mount services: Mount is to make a group of files in a file system structure accessible to a user or user group.

Mount operation: mount(remotehost, remotedirectory, localdirectory)



Client with two remotely mounted file stores. The nodes people and users in file systems at Server 1 and Server 2 are mounted over nodes students and staff in Client's local file store. The meaning of this is that programs running at Client can access files at Server 1 and Server 2 by using pathnames such as /usr/students/jon and /usr/staff/ann.

Remote file systems may be hard-mounted or soft-mounted in a client computer.
When a user-level process accesses a file in a file system that is hard-mounted, the process is suspended until the request can be completed, and if the remote host is unavailable for any reason the NFS client module continues to retry the request until it is satisfied. Thus in the case of a server failure, user-level processes are suspended until the server restarts and then they continue just as though there had been no failure. But if the relevant file system is soft-mounted, the NFS client module returns a failure indication to user-level processes after a small number of retries.

Pathname translation : UNIX file systems translate multi-part file pathnames to i-node references in a step-by-step process. In NFS, pathnames cannot be translated at a server, because the name may cross a 'mount point' at the client – directories holding different parts of a multi-part name may reside in filesystems at different servers. So pathnames are parsed, and their translation is performed in an iterative manner by the client. Each part of a name that refers to a emote-mounted directory is translated to a file handle using a separate lookup request to the remote server.

Automounter : The automounter was added to the UNIX implementation of NFS in order to mount a remote directory dynamically whenever an 'empty' mount point is referenced by a client. Caching in both the client and the server computer are indispensable features of NFS implementations in order to achieve adequate performance.

Server caching: NFS servers use the cache at the server machine just as it is used for other file accesses. The use of the server's cache to hold recently read disk blocks does not raise any consistency problems; but when a server performs write operations, extra measures are needed to ensure that clients can be confident that the results of the write operations are persistent, even when server crashes occur. The write operation offers two options for this :

1. Data in write operations received from clients is stored in the memory cache at the server and written to disk before a reply is sent to the client. This is called write-through caching. The client can be sure that the data is stored persistently as soon as the reply has been received.

2. Data in write operations is stored only in the memory cache. It will be written to disk when a commit operation is received for the relevant file. The client can be sure that the data is persistently stored only when a reply to a commit operation for the relevant file has been received. Standard NFS clients use this mode of operation, issuing a commit whenever a file that was open for writing is closed.

Client caching • The NFS client module caches the results of read, write, getattr, lookup and readdir operations in order to reduce the number of requests transmitted to servers. A timestamp-based method is used to validate cached blocks before they are used. Each data or metadata item in the cache is tagged with two timestamps:

-Tc is the time when the cache entry was last validated.

-Tm is the time when the block was last modified at the server.

Securing NFS with Kerberos •

The security of NFS implementations has been strengthened by the use of the Kerberos scheme to authenticate clients. In the original standard implementation of NFS, the user's identity is included in each request in the form of an unencrypted numeric identifier. NFS does not take any further steps to check the authenticity of the identifier supplied. This implies a high degree of trust in the integrity of the client computer and its software by NFS, whereas the aim of Kerberos and other authentication-based security systems is to reduce to a minimum the range of components in which trust is assumed. Essentially, when NFS is used in a 'Kerberized' environment it should accept requests.

NAME

In a distributed system names are used to refer to a wide variety of resources such as computers, services, remote objects, and files as well as users. Names are used for identification as well as for describing attributes.

Names = strings used to identify objects (files, computers, people, processes, objects)

● **Textual names:** Human-readable names are file names such as /etc/passwd, URLs such as http://www.cdk5.net/ and Internet domain names such as www.cdk5.net .
● **Numeric addresses:** , e.g. 193.206.186.100 (IP host address)
● **Object identifiers** : object's address: a value that identifies the location of the object rather than the object itself.

For many purposes, names are preferable to identifiers, because the binding of the named resource to a physical location is deferred and can be changed and also they are more meaningful to users.

Currently, different name systems are used for each type of resource:
● file-pathname
● process-process id
● Port-port number

Uniform Resource Identifiers (URI) offer a general solution for any type of resource. There two main classes:
-URL: Uniform Resource Locator
• typed by the protocol field (http, ftp, nfs, etc.)
• part of the name is service-specific

• resources cannot be moved between domains

-URN: Uniform Resource Name
• requires a universal resource name lookup service

Format: urn: <nameSpace>:<name-within namespace>
- Examples:
a) urn:ISBN:021-61918-0
b) urn:dcs.qmul.ac.uk :TR2007-5

a)send a request to nearest ISBN-lookup service - it would return whatever attributes of a book are required by the requester
b)send a request to the urn lookup service at dcs.qmul.ac.uk
- it would return a url for the relevant document

Name services

Name System (or Service) an Internet service that translates textual names and attributes for objects.

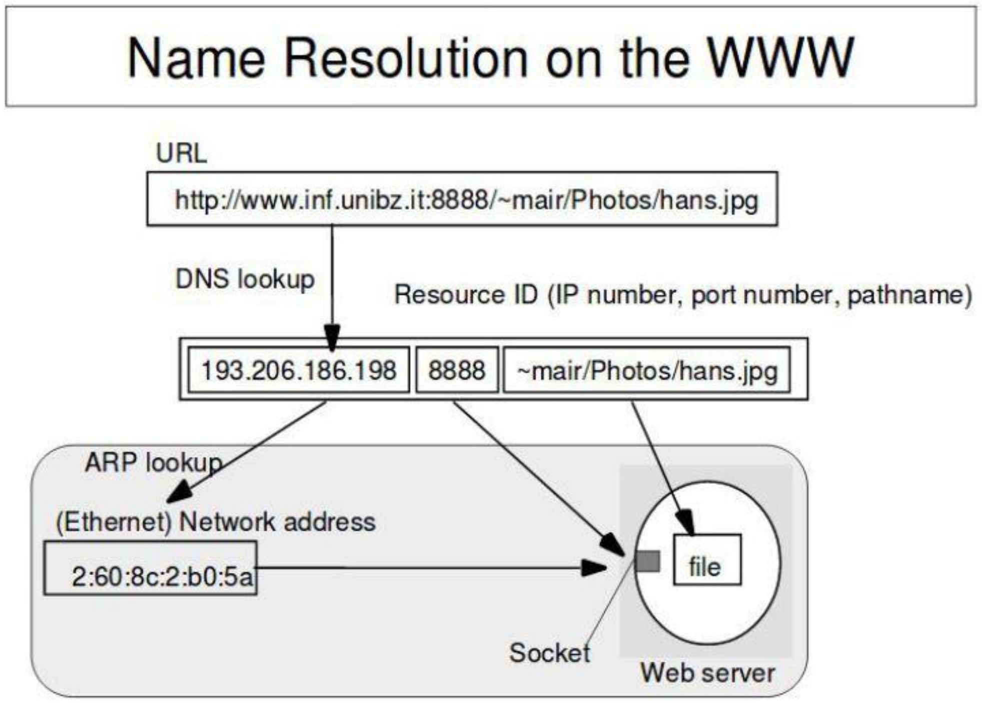
Examples of Name Services
● File system: maps file name to file
● RMI registry:binds remote objects to symbolic names
● DNS: maps domain names to IP addresses
● X.500/LDAP directory service: maps person's name to email address, phone number

Every time you use a domain name, therefore, a DNS service must translate the name into the corresponding IP address. For example, the domain name www.example.com might translate to 198.105.232.4.

Name management is separated from other services largely because of the openness of distributed systems, which brings the following motivations:

● Unification: It is often convenient for resources managed by different services to use the same naming scheme. URIs are a good example of this.

● Integration: It is not always possible to predict the scope of sharing in a distributed system. It may become necessary to share and therefore name resources that were


Name Resolution on the WWW
URL
http://www.inf.unibz.it:8888/~mair/Photos/hans.jpg
DNS lookup
Resource ID (IP number, port number, pathname)
193.206.186.198
8888
~mair/Photos/hans.jpg
ARP lookup
(Ethernet) Network address
2:60:8c:2:b0:5a
file
Socket
Web server

created in different administrative domains.

Three Design issues,
● Name spaces
● Name Resolution
● The domain name system

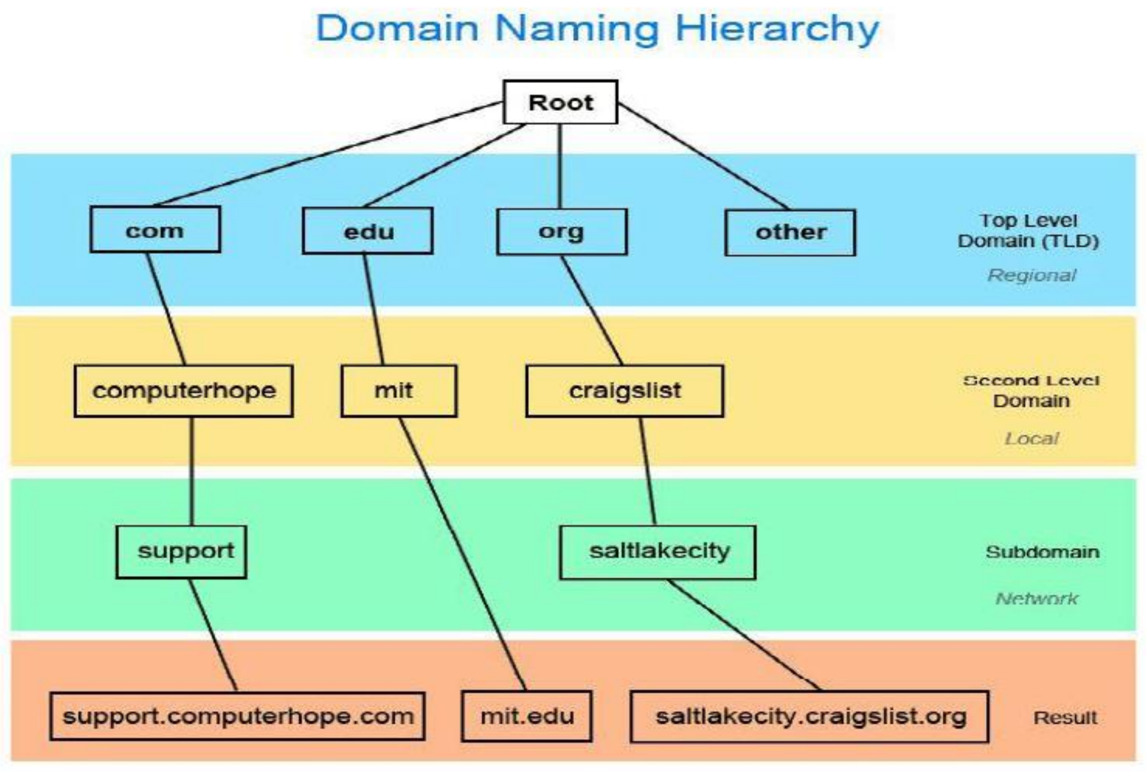
NAMESPACE AND DOMAIN NAME SYSTEM

DNS is the name service provided by the Internet for TCP/IP networks. DNS is broken up into domains, a logical organization of computers that exist in a larger network. Names may have an internal structure that represents their position,

● Hierarchy Namespace
● Flat name space: Single global context and naming authority for all names. So difficult to manage

Naming domains • A naming domain is a name space for which there exists a single overall administrative authority responsible for assigning names within it.eg.- .net, .com. The domains exist at different levels and connect in a hierarchy that resembles the root structure of a tree. Each domain extends from the node above it, beginning at the top with the root-level domain.

Under the root-level domain are the top-level domains, under those are the second-level domains, and on down into subdomains. DNS namespace identifies the structure of the domains that combine to form a complete domain name. For example, in the domain name sub.secondary.com, "com" is the top-level domain, "secondary" identifies the secondary domain name (commonly a site hosted by an organization and/or business), and "sub" identifies a subdomain within the larger network. This entire DNS domain structure is called the DNS namespace. The name assigned to a domain or computer relates to its position in the namespace.



Aliases • An alias is a name defined to denote the same information as another name. eg.- http://espn.go.com/ and http://www.espn.com

Combining and customizing name spaces • The DNS provides a global and homogeneous name space in which a given name refers to the same entity, no matter which process on which computer looks up the name.

Merging: how to merge the entire UNIX file systems of two (or more) computers called red and blue. Each computer has its own root, with overlapping file names. For example, /etc/passwd refers to one file on red and a different file on blue. The obvious way to merge the file systems is to replace each computer's root with a'super root' and mount each computer's file system in this super root, say as /red and /blue. Users and programs can then refer to /red/etc/passwd and /blue/etc/passwd.

Domain name system(DNS)

Domain names

• The DNS is designed for use in multiple implementations, each of which may have its own name space. In practice, however, only one is in widespread use, and that is the one used for naming across the Internet. The Internet DNS name space is partitioned both organizationally and according to geography. The names are written with the highest-level domain on the right. The original top-level organizational domains (also called generic domains) in use across the Internet were:

● com - Commercial organizations
● edu- Universities and other educational institutions
● gov- governmental agencies
● mil- military organizations
● net- Major network support centres
● org- Organizations
● int- International organizations

In addition, every country has its own domains:
● us-United States
● uk-United Kingdom
● fr-France

DNS queries: The Internet DNS is primarily used for simple host name resolution and for looking up electronic mail hosts, as follows

● Host name resolution:when a web browser is given a URL containing the domain name www.dcs.qmul.ac.uk, it makes a DNS enquiry and obtains the corresponding IP address.

● Mail host location: Electronic mail software uses the DNS to resolve domain names into the IP addresses of mail hosts. For example, when the address tom@dcs.rnx.ac.uk is to be resolved, the DNS is queried with the address dcs.rnx.ac.uk and the type designation ‘mail’. It returns a list of domain names of hosts that can accept mail for dcs.rnx.ac.uk Some other types of query that are implemented in some installations but are less frequently used than those just given are:

● Reverse resolution: Some software requires a domain name to be returned given an IP address.

● Host information: The DNS can store the machine architecture type and operating system with the domain names of hosts.

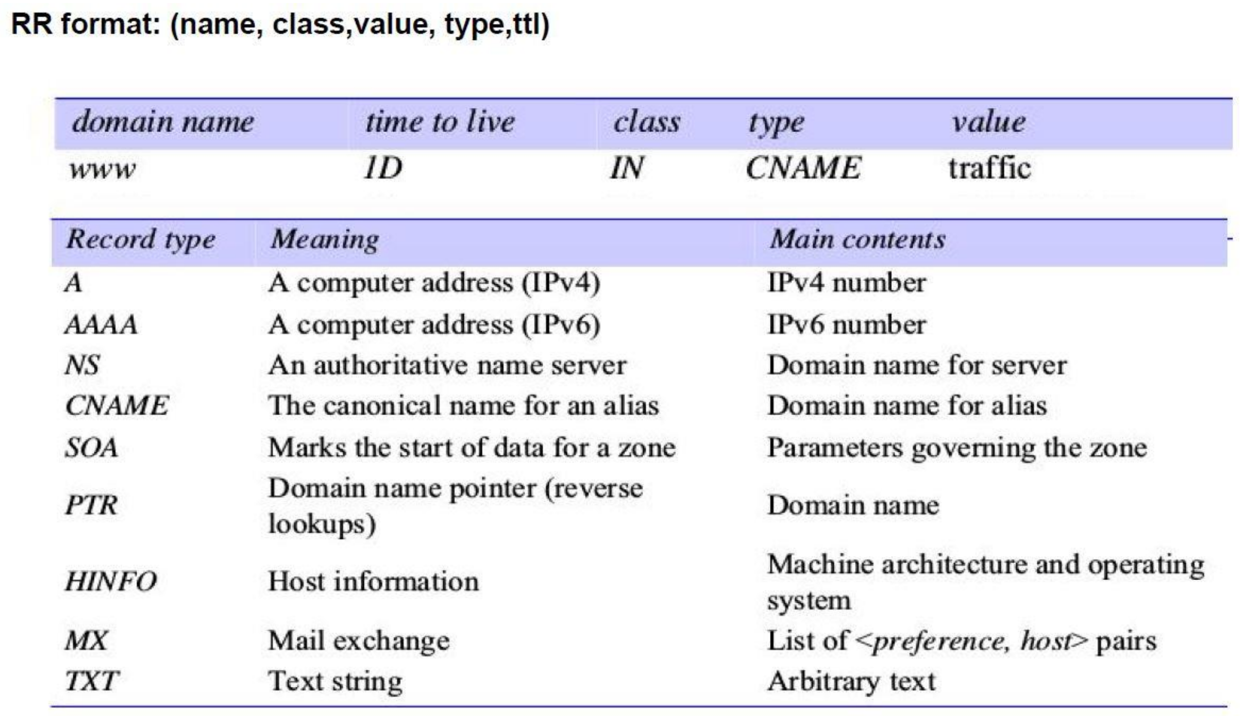
DNS name servers: A ll host names and addresses in one large master file stored on one central host. Every domain name, which is a part of the DNS system, has several DNS settings, also known as DNS records. In order for these DNS records to be kept in order, the DNS zone was created. Every zone must have at least two name servers

● exactly one master (= primary) server: contains the only writable copy of the "zone file"
● one or more secondary (= slave) servers: copies its zone file from the master

DNS server caching the lookup result for a limited time, known as its Time To Live (**TTL**), ranging from a few minutes to a few days. People managing a DNS server can configure its TTL, so TTL values will vary across the Internet.

DNS resource records:

DNS holds resource records (RR).



RR format: (name, class,value, type,ttl)

domain name	*time to live*	*class*	*type*	*value*
www	*1D*	*IN*	*CNAME*	traffic

Record type	*Meaning*	*Main contents*
A	A computer address (IPv4)	IPv4 number
AAAA	A computer address (IPv6)	IPv6 number
NS	An authoritative name server	Domain name for server
CNAME	The canonical name for an alias	Domain name for alias
SOA	Marks the start of data for a zone	Parameters governing the zone
PTR	Domain name pointer (reverse lookups)	Domain name
HINFO	Host information	Machine architecture and operating system
MX	Mail exchange	List of <*preference, host*> pairs
TXT	Text string	Arbitrary text

BIND *or Berkeley Internet Name Domain* , is most widely used Open source software that implements DNS protocols for internet, which provides us ability to implement IP to domain name conversion & vice-versa .

Name resolution (Name resolver)

DNS name resolution is nothing but resolving host names, such as www.nixcraft.com, to their corresponding IP addresses. DNS works as the phone book" for the Internet by translating hostname into IP address or vise versa.

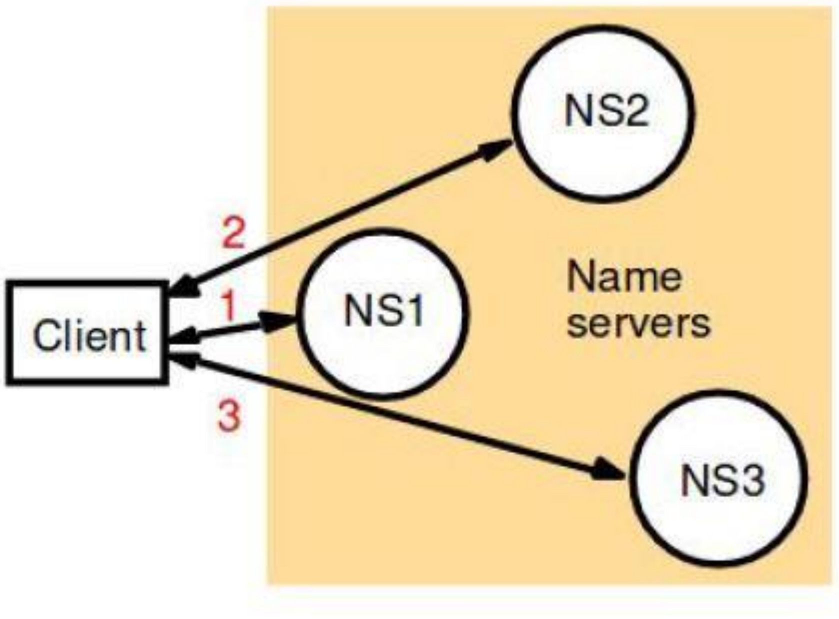
Name servers and navigation

DNS, that stores a very large database and is used by a large population will not store all of its naming information on a single server computer. The process of locating naming data from more than one name server in order to resolve a name is called navigation.

● iterative navigation
● Multicast navigation
● server control navigation
    -Recursive navigation
    -Nonrecursive navigation

Iterative navigation:

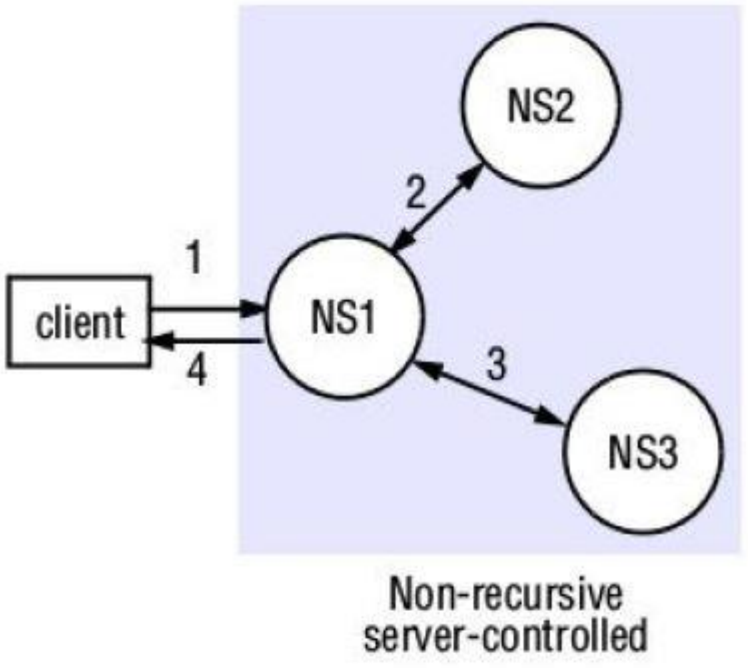
To resolve a name, a client presents the name to the local name server, which attempts to resolve it. If the local name server has the name, it returns the result immediately. If it does not, it will suggest another server that will be able to help. DNS supports the model known as iterative navigation. Resolution continues until name resolved or name found to be unbound.



Example: If you enter www.example.com in the browser, the operating system's resolver will send this query for the record to the a DNS server NS1. On receiving the query, it will look through its tables(cache) to find the IP address for the domain www.example.com. But if it does not have the entry then NS1 will reply back to client with a referral to another servers. Then operating system resolver, will send the query to NS2. And it continues the process until it resolved.

Multicast navigation:
In multicast navigation, a client multicasts the name to be resolved and the required object type to the group of name servers. Only the server that holds the named attributes responds to the request.

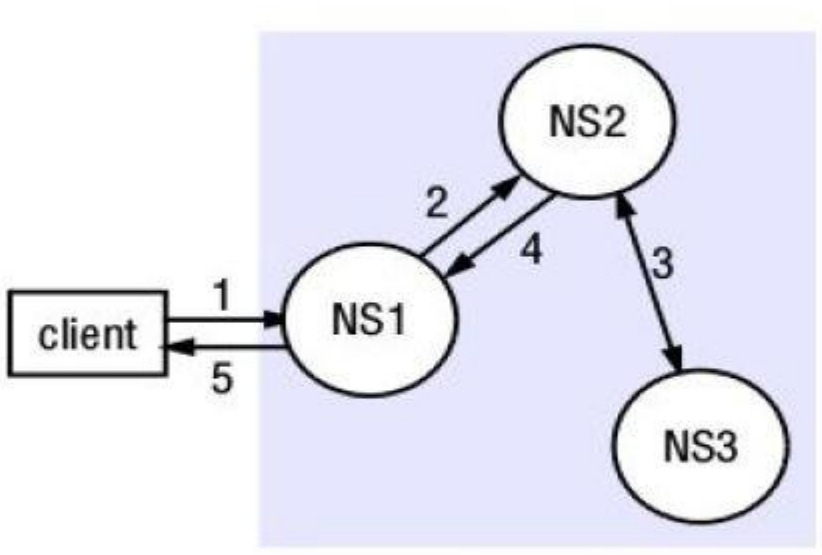
Non-recursive server-controlled navigation
Under non-recursive server-controlled navigation, any name server may be chosen by the client. This server communicates by multicast or iteratively with its peers in the style described above, as though it were a client.



Example: If you enter www.example.com in the browser, the operating system's resolver will send this query for the record to the a DNS server NS1. On receiving the query, it will look through its tables(cache) to find the IP address for the domain www.example.com. But if it does not have the entry then NS1 contacts peers if it cannot resolve name itself by multicast or iteratively by direct contact. Answer for the query will send back to client by NS1.

Recursive server-controlled navigation:

Under recursive server-controlled navigation, the client once more contacts a single server. If this server does not store the name, the server contacts a peer storing a (larger) prefix of the name, which in turn attempts to resolve it. This procedure continues recursively until the name is resolved.



Recursive
server-controlled

Example: If you enter www.example.com in the browser, the operating system's resolver will send this query for the record to the a DNS server NS1. On receiving the query, it will look through its tables(cache) to find the IP address for the domain www.example.com. But if it does not have the entry then NS1 contacts NS2. If NS2 does not have the entry then send the query to NS3. Answer for the query will send back to client by NS1. This procedure continues recursively until the name is resolved. Answer for the query will send back to client by NS1.

Caching • In DNS and other name services, client name resolution software and servers maintain a cache of the results of previous name resolutions. When a client requests a name lookup, the name resolution software consults its cache.

Directory services

A directory service is the collection of software and processes that store information (name,attribute). An example of a directory service is the Domain Name System (DNS), which is provided by DNS servers. A DNS server stores the mappings of computer host names and other forms of domain name to IP addresses. A DNS client sends questions to a DNS server about these mappings (e.g. what is the IP address of test.example.com?). Thus, all of the computing resources (hosts) become clients of the DNS server. The mapping of host names enables users of the computing resources to locate computers on a network, using host names rather than complex numerical IP addresses.

Directory services are sometimes called yellow pages services, and conventional name services are correspondingly called white pages services, in an analogy with the traditional types of telephone directory. Directory services are also sometimes known as attribute-based name services, eg. X.500, LDAP, MS Active Directory Services. However, any organization that plans to base its applications on web services will find it more convenient to use a directory service to make these services available to clients. This is the purpose of the Universal Description, Discovery and Integration service (UDDI). UDDI provides both white pages and yellow pages services (a white pages service by name or a yellow pages service by attribute(IP)).

Discovery service: a special case of a directory service for services provided by devices in a spontaneous networking environment

● automatically updated as the network configuration changes
● discovers services required by a client (who may be mobile) within the current scope, for example, to find the most suitable printing service for image files after arriving at a hotel

Case study: The Global Name Service

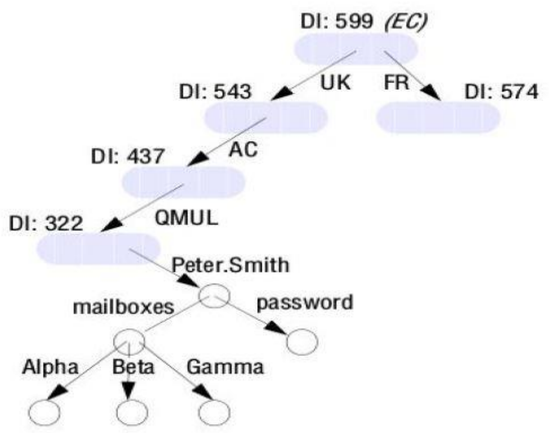
Designed and implemented by Lampson and colleagues at the DEC Systems Research Center (1986). Mainly used to merge two more name servers.

● Also Provide facilities for resource location, email addressing and authentication

The GNS manages a naming database that is composed of a tree of directories holding names and values. Directories are named by multi-part pathnames referred to a root, or relative to a working directory, much like file names in a UNIX file system. Each directory is also assigned an integer, which serves as a unique *directory identifier* (DI) and EC is directory. A directory contains a list of names and references. The values stored at the leaves of the directory tree are organized into *value trees* , so that the attributes associated with names can be structured values.

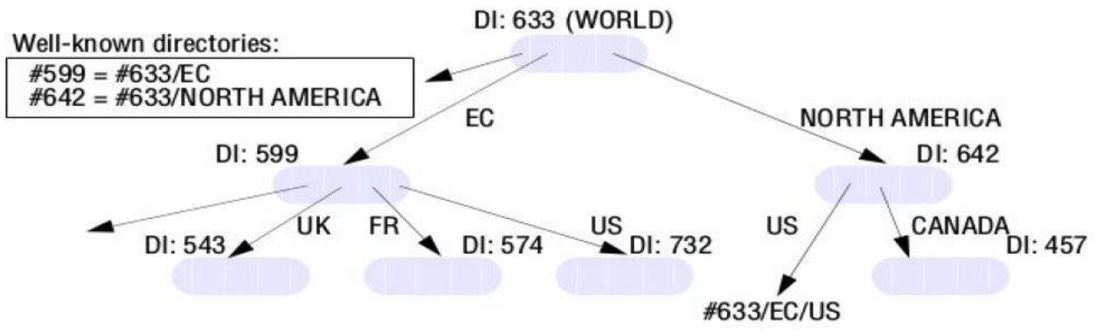
Names in the GNS have two parts: < *directory name* , *value name* >. The first part identifies a directory; the second refers to a value tree, or some portion of a value tree.

Mechanism -> add a new root node and make the exiting root node its children



Eg.The attributes of a user Peter.Smith in the directory QMUL would be stored in the value tree named <EC/UK/AC/QMUL, Peter.Smith>. The value tree includes a password, which can be referenced as <EC/UK/AC/QMUL, Peter.Smith/password>, and several mail addresses, each of which would be listed in the value tree as a single node with the name <EC/UK/AC/QMUL, Peter.Smith/mailboxes>.

At the level of clients and administrators, growth is accommodated through extension of the directory tree in the usual manner. But we may wish to integrate the naming trees of two previously separate GNS services.For example, how could we integrate the database rooted at the EC directory shown in above Figure with another database for NORTH AMERICA.Below Figure shows a new root, WORLD, introduced above the existing roots of the two trees to be merged. Here only Problem is Existing names need to be changed.

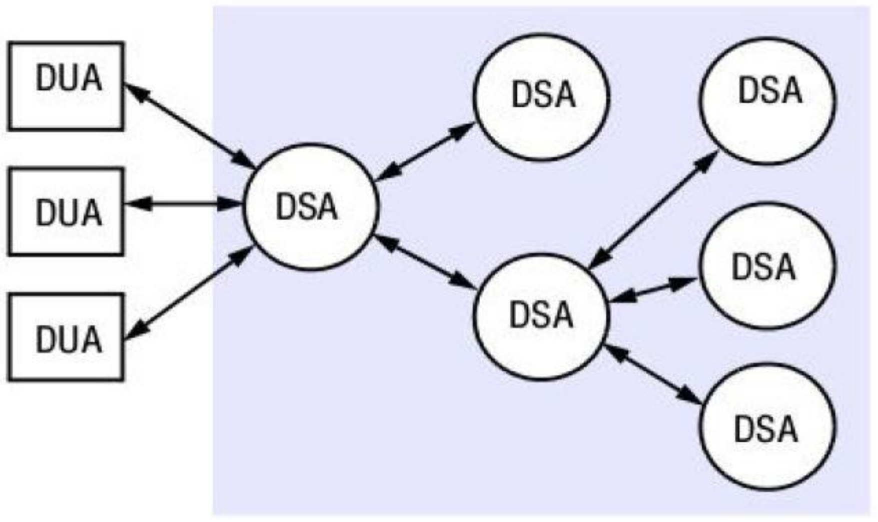


For example, </UK/AC/QMUL, Peter.Smith> is a name used by clients before integration. The root it refers to is EC, not WORLD. EC and NORTH AMERICA are working roots – initial contexts against which names beginning with the root '/' are to be looked up.The existence of unique directory identifiers can be used to solve this problem. The working root for each program must be identified as part of its execution environment (much as is done for a program's working directory). When a client in the European Community uses a name of the form </UK/AC/QMUL, Peter.Smith>, its local user agent, which is aware of the working root, prefixes the directory identifier EC(#599), thus producing the name <#599/UK/AC/QMUL, Peter.Smith>.

Case study: The X.500 Directory Service

X.500 is a standard for directory services developed by the *International*

Telecommunications Union (ITU), the most recent version of which was published in 1993. It uses a distributed approach to implement a global directory service. Such a directory is sometimes called a global White Pages directory. The X.500 directory is organized under a common "root" directory in a "tree" hierarchy of: country,organization, organizational unit, and person.



In X.500, each local directory is called a Directory System Agent (DSA). A DSA can represent one organization or a group of organizations. The X.500 name tree is called then Directory Information Tree (DIT), and the entire directory structure including the data associated with the nodes, is called the Directory Information Base (DIB).

The user interface program for access to one or more DSAs is a Directory User Agent (DUA). The University of Michigan is one of a number of universities that use X.500 as a way to route e-mail as well as to provide name lookup, using the Lightweight Directory Access Protocol (LDAP).

MODULE 5

TRANSACTIONS

A transaction is a single logical unit of work which accesses and possibly modifies the contents of a database. Transactions access data using read and write operations. Let's take an example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.

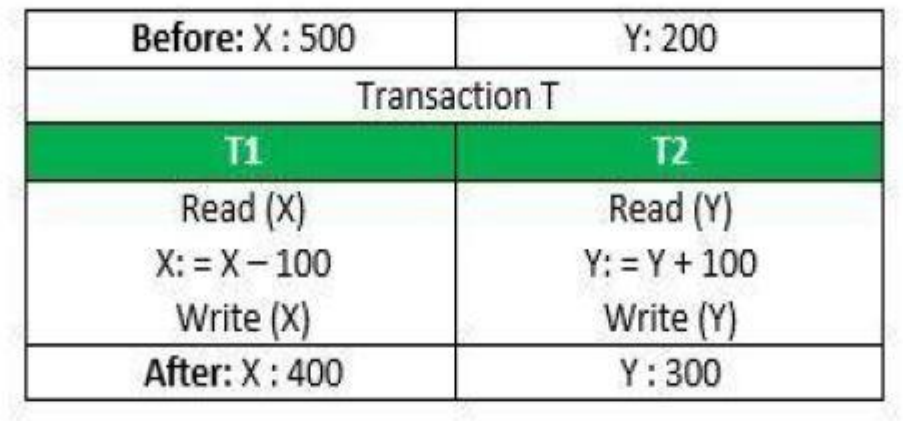
A's Account
Open_Account(A)
Old_Balance = A.balance
New_Balance = Old_Balance - 500
A.balance = New_Balance
Close_Account(A)

B's Account
Open_Account(B)
Old_Balance = B.balance
New_Balance = Old_Balance + 500
B.balance = New_Balance
Close_Account(B)

In order to maintain consistency in a database, before and after transaction, certain properties are followed. These are called **ACID** properties.

● **Atomicity:**
By this, we mean that either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves following two operations. Atomicity is also known as the 'All or nothing rule'.

—Abort: If a transaction aborts, changes made to database are not visible.
—Commit: If a transaction commits, changes made are visible.
Consider the following transaction T consisting of T1 and T2: Transfer of 100 from account X to account Y.



Before: X : 500	Y: 200
Transaction T	
T1	T2
Read (X) X: = X − 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

If the transaction fails after completion of T1 but before completion of T2.(say, after write(X) but before write(Y)), then amount has been deducted from X but not added to Y. This results in an inconsistent database state. Therefore, the transaction must be executed in entirety in order to ensure correctness of database state.

● **Consistency:**

This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to correctness of a database. Referring to the example above,

The total amount before and after the transaction must be maintained.
Total before T occurs = 500 + 200 = 700.
Total after T occurs = 400 + 300 = 700.

Therefore, database is consistent. Inconsistency occurs in case T1 completes but T2 fails. As a result T is incomplete.

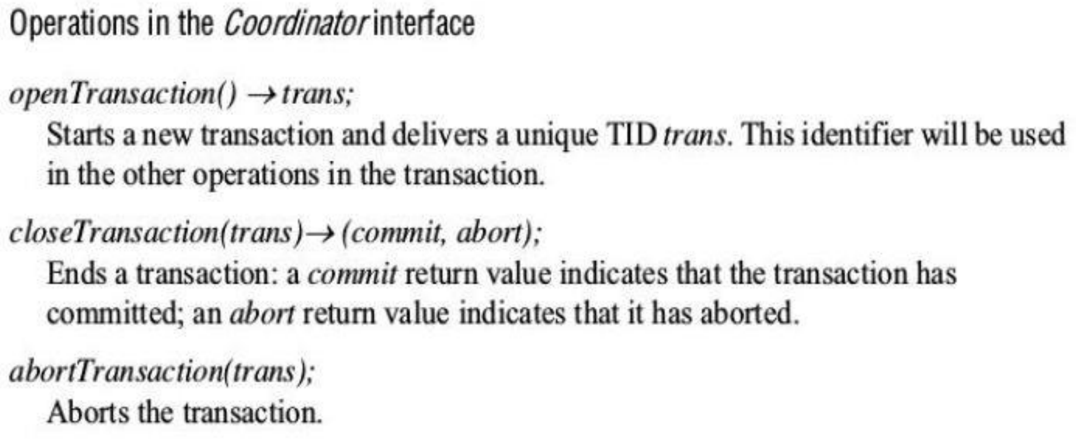
● **Isolation:**

This property ensures that multiple transactions can occur concurrently without leading to inconsistency of database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.

● **Durability:**
This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even is system failure occurs. These updates now become permanent and are stored in a non-volatile memory.
The effects of the transaction, thus, are never lost Transaction sequence must continue until:

● COMMIT statement is reached
● ROLLBACK statement is reached
● End of program is reached
● Program is abnormally terminated

SQL statements that provide transaction support,
● COMMIT
● ROLLBACK
Transaction capabilities can be added to servers of recoverable objects. Each transaction is created and managed by a coordinator.



Operations in the *Coordinator* interface

openTransaction() → trans;
Starts a new transaction and delivers a unique TID *trans*. This identifier will be used in the other operations in the transaction.

closeTransaction(trans)→ (commit, abort);
Ends a transaction: a *commit* return value indicates that the transaction has committed; an *abort* return value indicates that it has aborted.

abortTransaction(trans);
Aborts the transaction.

Service actions related to process crashes • If a server process crashes unexpectedly, it is eventually replaced. The new server process aborts any uncommitted transactions and uses a recovery procedure to restore the values of the objects to the values produced by the most recently committed transaction. To deal with a client that crashes unexpectedly during a transaction, servers can give each transaction an expiry time and abort any transaction that has not completed before its expiry time.

Client actions related to server process crashes • If a server crashes while a transaction is in progress, the client will become aware of this when one of the operations returns an exception after a timeout. If a server crashes and is then replaced during the progress of a transaction, the transaction will no longer be valid and the client must be informed via an exception to the next operation. In either case, the client must then formulate a plan, possibly in consultation with the human user, for the completion or abandonment of the task of which the transaction was a part.

Concurrency control

Concurrency controlling techniques ensure that multiple transactions are executed simultaneously while maintaining the ACID properties of the transactions and serializability in the schedules. **Serializability** is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database. The various approaches for concurrency control,

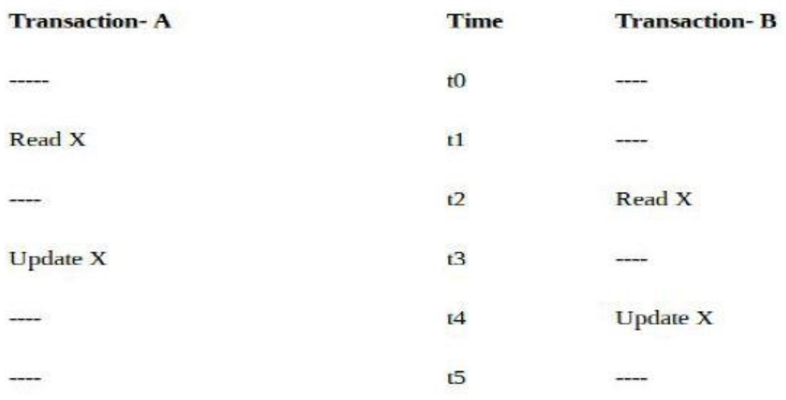
● Lock based concurrency control
● Timestamp concurrency control
● Optimistic concurrency control

Simultaneous execution of transactions over a shared database can create several data integrity and consistency problems.

● **Lost Update:**
● **Dirty Read Problems**
● **Inconsistent Retrievals**

Lost Update: This problem occurs when two transactions that access the same database items, have their operations in a way that makes the value of some database item incorrect.
In other words, if transactions T1 and T2 both read a record and then update it, the effects of the first update will be overwritten by the second update.

Example:
Consider the situation given in figure that shows operations performed by two transactions, Transaction- A and Transaction- B with respect to time.



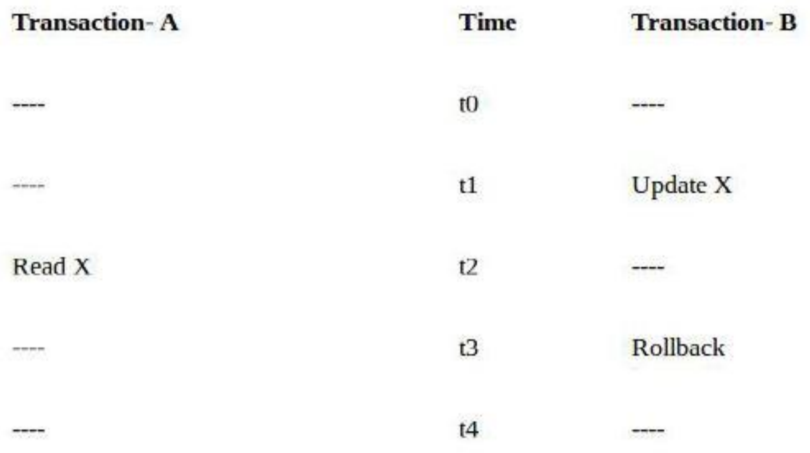
Transaction- A	Time	Transaction- B
-----	t0	----
Read X	t1	----
----	t2	Read X
Update X	t3	----
----	t4	Update X
----	t5	----

At time t1 , Transactions-A reads value of X.
At time t2 , Transactions-B reads value of X.
At time t3,Transactions-A writes value of X on the basis of the value seen at time t1.
At time t4,Transactions-B writes value of X on the basis of the value seen at time t2.

So,update of Transactions-A is lost at time t4,because Transactions-B overwrites it without looking at its current value.
Such type of problem is referred as the Update Lost Problem, as update made by one transaction is lost here

Dirty Read Problems: This problem occurs when one transaction reads changes the value while the other reads the value before committing or rolling back by the first transaction.

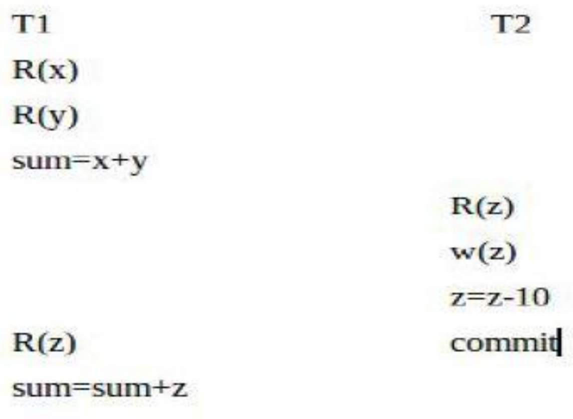
Example:
Consider the situation given in figure :



Transaction- A	Time	Transaction- B
----	t0	----
----	t1	Update X
Read X	t2	----
----	t3	Rollback
----	t4	----

At time t1 , Transactions-B writes value of X.
At time t2 , Transactions-A reads value of X.
At time t3 , Transactions-B rollbacks.So,it changes the value of X back to that of prior to t1.
So,Transaction-A now has value which has never become part of the stable database. Such type of problem is referred as the Dirty Read Problem, as one transaction reads a dirty value which has not been committed.

Inconsistent Retrievals(Unrepeatable Read Problem) : occurs when a transaction calculates some summary (aggregate) function over a set of data while other transactions are updating the data. example: let x=10, y=20, z=20



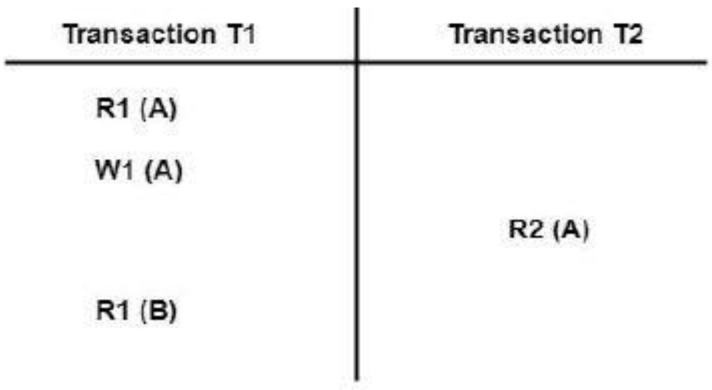
T1	T2
R(x)	
R(y)	
sum=x+y	
	R(z)
	w(z)
	z=z-10
R(z)	commit
sum=sum+z	

Serializability : Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database(Interleaved execution of transactions yields the same results as the serial execution of the transactions.)

Conflicting operations: Two operations are called as conflicting operations if all the following conditions hold true for them-

● Both the operations belong to different transactions
● Both the operations are on same data item
● Read – Read (No conflict)
● Read – Write (or Write – Read)Conflict!
● Write – Write (Conflict)

Example-
Consider the following schedule-



Transaction T1	Transaction T2
R1 (A)	
W1 (A)	
	R2 (A)
R1 (B)	

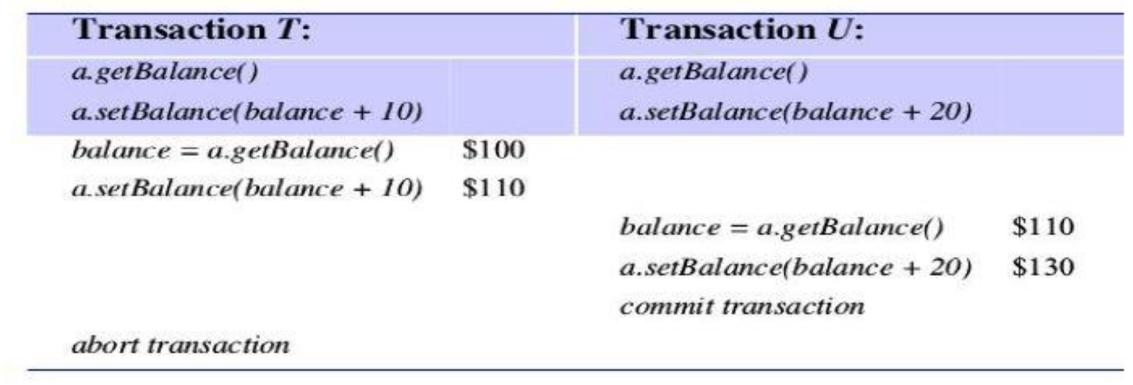
In this schedule, W1 (A) and R2 (A) are called as conflicting operations because all the above conditions hold true for them.

Recoverability from aborts

Servers must record all the effects of committed transactions and none of the effects of aborted transactions. This section illustrates two problems associated with aborting transactions,

● dirty reads
● premature

Dirty reads • The isolation property of transactions requires that transactions do not see the uncommitted state of other transactions. The ‘dirty read’ problem is caused by the interaction between a read operation in one transaction and an earlier write operation in another transaction on the same object



Transaction *T*:		**Transaction *U*:**	
a.getBalance()		*a.getBalance()*	
a.setBalance(balance + 10)		*a.setBalance(balance + 20)*	
balance = a.getBalance()	$100		
a.setBalance(balance + 10)	$110		
		balance = a.getBalance()	$110
		a.setBalance(balance + 20)	$130
		commit transaction	
abort transaction			

T2 sees result update by T1 on account A
T2 performs its own update on A & then commits.
T1 aborts -> T2 has seen a “transient” value
T2 is not recoverable

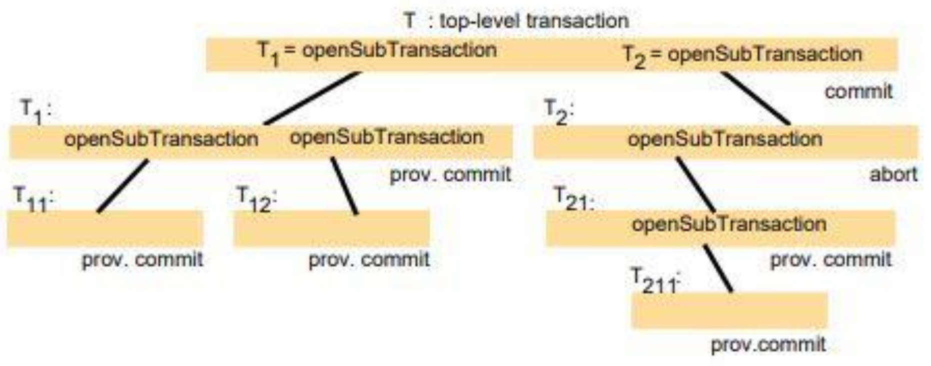
The failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is known as a **cascading schedule** or **cascading rollback** or **cascading abort** .

Premature writes:
Assume server implements abort by maintaining the “before” image of all update operations
T1 & T2 both updates account A
T1 completes its work before T2
If T1 commits & T2 aborts, the balance of A is correct
If T1 aborts & T2 commits, the “before” image that is restored corresponds to the balance of A before T2

For recoverability:
A commit is delayed until after the commitment of any other transaction whose state has been observed (Tx’s should be delayed until earlier Tx’s that update the Same objects have been either committed or aborted.)

NESTED TRANSACTIONS

The nested transaction is a transaction that is created inside another transaction. A *nested transaction* is used to provide a transactional guarantee for a subset of operations performed within the scope of a larger transaction. Doing this allows you to commit and abort the subset of operations independently of the larger transaction.



2

Nested transactions have the following main advantages:

1. Subtransactions at one level (and their descendants) may run concurrently with other subtransactions at the same level in the hierarchy.
2. Subtransactions can commit or abort independently.

The rules to the usage of a nested transaction are as follows:

● While the nested (child) transaction is active, the parent transaction may not perform any operations other than to commit or abort, or to create more child transactions.

● Committing a nested transaction has no effect on the state of the parent transaction. The parent transaction is still uncommitted.

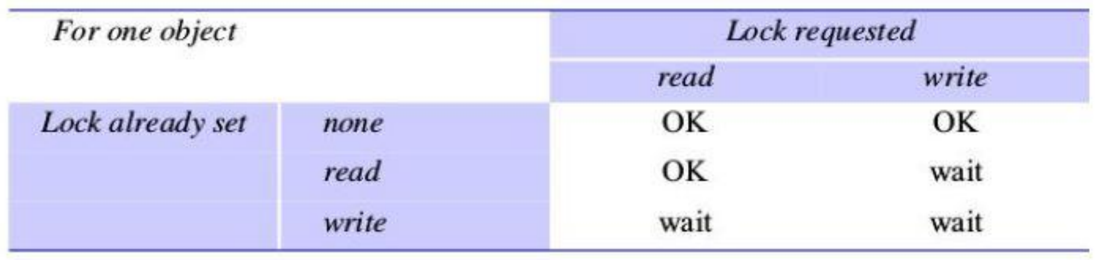
● Likewise, aborting the nested transaction has no effect on the state of the parent transaction.

● If the parent aborts, then the child transactions abort as well. If the parent commits, then whatever modifications have been performed by the child transactions are also committed.

LOCKS

Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds ,

● Binary Locks − A lock on a data item can be in two states; it is either locked or unlocked.
● Shared/exclusive − If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.


Lock compatibility

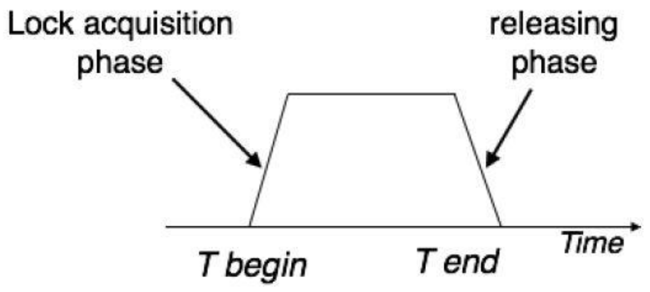
For one object		*Lock requested*	
		read	*write*
Lock already set	*none*	OK	OK
	read	OK	wait
	write	wait	wait

Types of lock protocols,

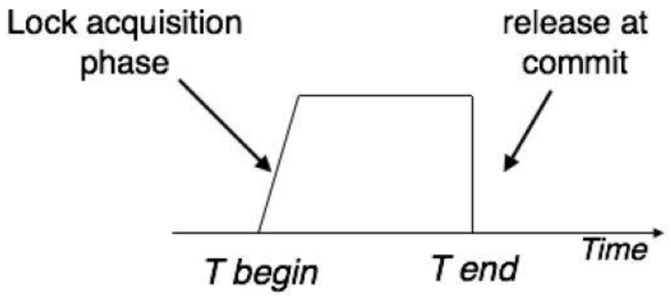
● Single phase locking
● Two phase locking- 2PL
● Strict two phase locking

Single phase : lock-based protocols allow transactions to obtain a lock on every object before a operation is performed. Transactions may unlock the data item after completing the operation.

Two phase: This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks (Growing phase). Third phase starts, the transaction cannot demand any new locks; it only releases the acquired locks (Shrinking phase).



Strict Two-Phase Locking: The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.



Use of locks in strict two-phase locking

1. When an operation accesses an object within a transaction:
(a)If the object is not already locked, it is locked and the operation proceeds.
(b)If the object has a conflicting lock set by another transaction, the transaction must wait until it is unlocked.
(c)If the object has a non-conflicting lock set by another transaction, the lock is shared and the operation proceeds.

2. When a transaction is committed or aborted, the server unlocks all objects it locked for the transaction

Lock implementation

The granting of locks will be implemented by a separate object in the server that we call the lock manager. The lock manager holds a set of locks. The methods of Lock are synchronized so that the threads attempting to acquire or release a lock will not interfere with one another. But, in addition, attempts to acquire the lock use the wait method whenever they have to wait for another thread to release it.

All requests to set locks and to release them on behalf of transactions are sent to an instance of LockManager:

• The setLock method's arguments specify the object that the given transaction wants to lock and the type of lock.
• The unLock method's argument specifies the transaction that is releasing its locks.

Locking rules for nested transactions

The aim of a locking scheme for nested transactions is to serialize access to objects so that,
● Each set of nested transactions is a single entity that must be prevented from observing the partial effects of any other set of nested transactions.
● Each transaction within a set of nested transactions must be prevented from observing the partial effects of the other transactions in the set.

The **first rule** is enforced by arranging that every lock that is acquired by a successful subtransaction is inherited by its parent when it completes. Inherited locks are also inherited by ancestors. The top-level transaction eventually inherits all of the locks that were acquired by successful subtransactions at any depth in a nested transaction.

The **second rule** is enforced as follows:

• Parent transactions are not allowed to run concurrently with their child transactions. This means that the child transaction temporarily acquires the lock from its parent for its duration.

• Subtransactions at the same level are allowed to run concurrently.

The following rules describe lock acquisition and release:

• For a subtransaction to acquire a read lock on an object, no other active transaction can have a write lock on that object.
• For a subtransaction to acquire a write lock on an object, no other active transaction can have a read or write lock on that object.
• When a subtransaction commits, its locks are inherited by its parent
• When a subtransaction aborts, its locks are discarded.

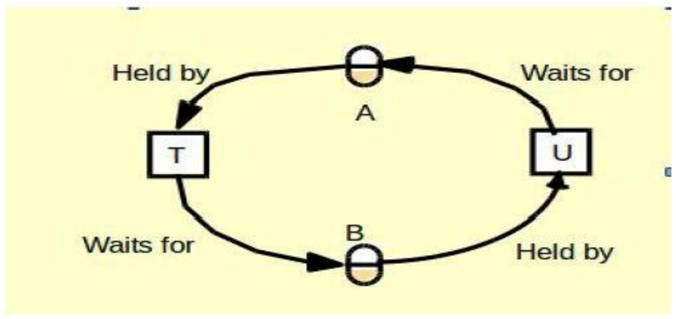
Deadlocks

Deadlock is a state in which each member of a group of transactions is waiting for some other member to release a lock.The disadvantage of Locking Deadlocks. For an example, two transaction T & U with two object A & B. T locks A and waits for U to release the locks on B. Other hand U locks B and waits for T to release the locks on A -> Deadlock happens.

Four necessary conditions of deadlock are,
● Mutual Exclusion
● Hold and waits
● Non preemption
● circular waits

Strategies to Fight Deadlock

Deadlock Prevention: Violate one of the necessary conditions for deadlock.
Deadlock Avoidance: Have transactions declare max resources they will request, but allow them to lock at any time (Banker's algorithm)
Deadlock detection • Deadlocks may be detected by finding cycles in the wait-for graph. Having detected a deadlock, a transaction must be selected for abortion to break the cycle.



Timeouts • Lock timeouts are a method for resolution of deadlocks that is commonly used. Each lock is given a limited period in which it is invulnerable. After this time, a lock becomes vulnerable. However, if any other transaction is waiting to access the object protected by a vulnerable lock, the lock is broken (that is, the object is unlocked) and the waiting transaction resumes.

OPTIMISTIC CONCURRENCY CONTROL

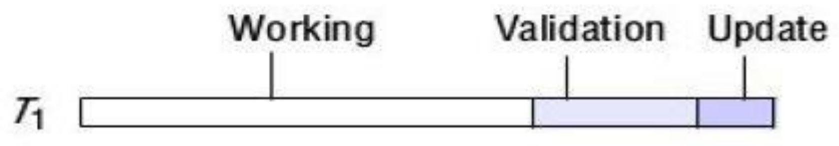
Concurrency control is a concept that is used to address conflicts with the simultaneous accessing or altering of data that can occur with a multi-user system. Optimistic concurrency control (OCC) is a concurrency control method applied to transactional systems. OCC assumes that multiple transactions can frequently complete without interfering with each other. Each transaction has the following phases:

● Working Phase
● Validation Phase
● Update Phase

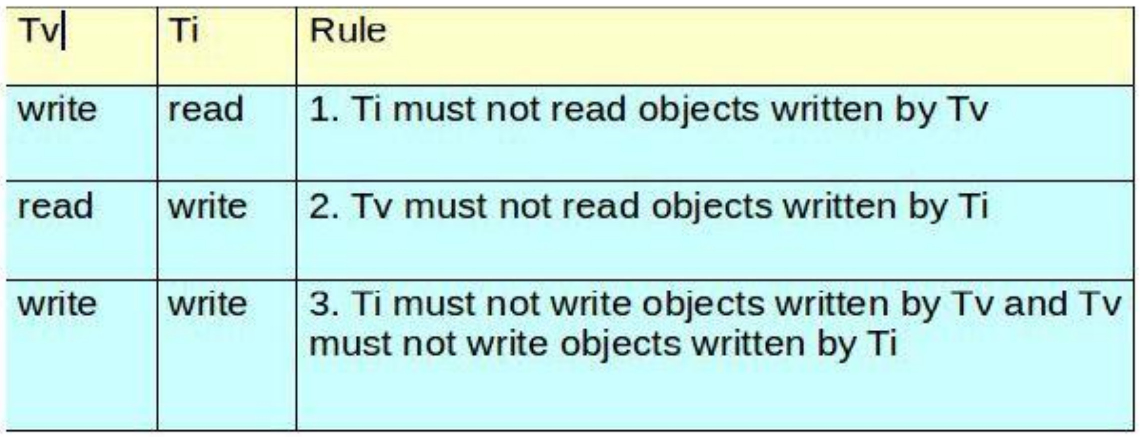
Working Phase: A transaction fetches data items to memory and performs operations upon them.

Validation Phase: When the closeTransaction request is received, the transaction is validated to establish whether or not its operations on objects conflict with operations of other transactions on the same objects.

Update Phase: If a transaction is validated, all of the changes recorded in its tentative versions are made permanent.



Validation uses the read-write conflict rules to ensure that the scheduling of a particular transaction is serially equivalent with respect to all other overlapping transactions – that is, any transactions that had not yet committed at the time this transaction started. To assist in performing validation, each transaction is assigned a transaction number when it enters the validation phase. If the transaction is validated and completes successfully, it retains this number; if it fails the validation checks and is aborted. The validation rules are,



Tv	Ti	Rule
write	read	1. Ti must not read objects written by Tv
read	write	2. Tv must not read objects written by Ti
write	write	3. Ti must not write objects written by Tv and Tv must not write objects written by Ti

Two types of Validation,
● Backward Validation
● Forward Validation

Backward Validation: In backward validation, the read set of the transaction being validated is compared with the write sets of other transactions that have already committed. Therefore, the only way to resolve any conflicts is to abort the transaction that is undergoing validation. In backward validation, transactions that have no read operations (only write operations) need not be checked.

Forward Validation: In forward validation of the transaction Tv, the write set of Tv is compared with the read sets of all overlapping active tractions- those that are still in their working phase (rule 1). Rule 2 is automatically fulfilled because that active transactions do not write until after Tv has completed.

Comparison of forward and backward validation

conflicts-

● Forward validation allows flexibility in the resolution of conflicts. (Two methods -Defer the validation until a later time when the conflicting transactions have finished or Abort the transaction being validated).

● Backward validation allows only one choice – to abort the transaction being validated

Read Set- The read sets of transactions are much larger than the write sets.,

● Backward validation compares a possibly large read set against the old write set and it has a overhead of storing old write sets.

● Forward validation checks a small write set against the read sets of active transactions and it has to allow for new transactions starting during the validation process.

MODULE 6

DISTRIBUTED MUTUAL EXCLUSION

Distributed processes often need to coordinate their activities. If a collection of processes share a resource or collection of resources, then often mutual exclusion is required to prevent interference and ensure consistency when accessing the resources.

The application-level protocol for executing a critical section is as follows:

● **enter()** : enter critical section – block if necessary

● **resourceAccesses()** : access shared resources in critical section

● **exit():** leave critical section – other processes may now enter

Our essential requirements for mutual exclusion are as follows:

● **ME1: (safety)** At most one process may execute in the critical section (CS) at a time.

● **ME2: (liveness)** Requests to enter and exit the critical section eventually succeed.

ME2 implies freedom from both deadlock and starvation. A deadlock would involve two or more of the processes becoming stuck indefinitely while attempting to enter or exit the critical section, by virtue of their mutual interdependence. But even without a deadlock, a poor algorithm might lead to starvation: the indefinite postponement of entry for a process that has requested it.

The absence of starvation is a fairness condition.

● **ME3: (ordering)** If one request to enter the CS happened-before another, then entry to the CS is granted in that order.

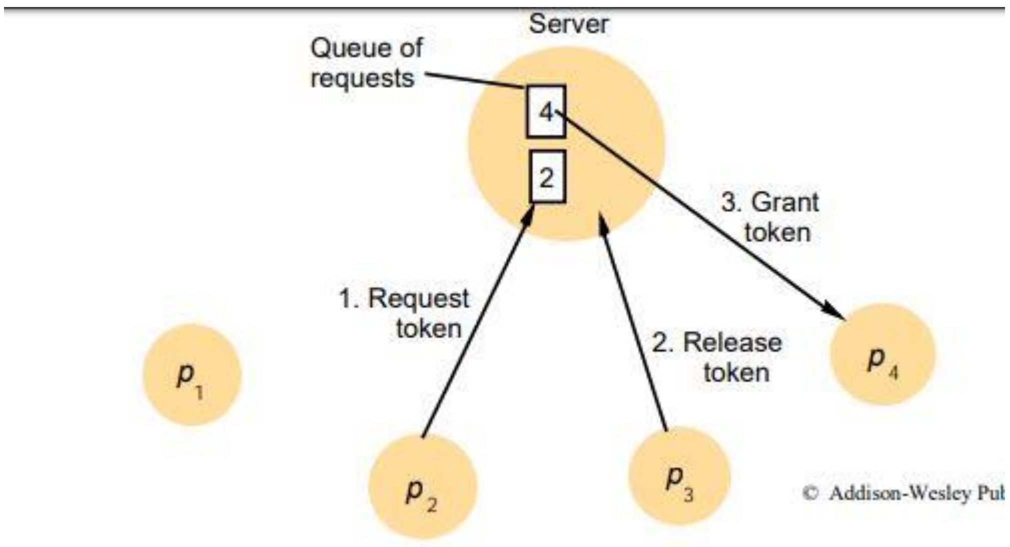
We evaluate the performance of algorithms for mutual exclusion according to the following criteria:

• the **bandwidth** consumed,
• the **client delay** incurred by a process at each entry and exit operation;
• the algorithm's effect upon the **throughput** of the system

Algorithms for mutual exclusion

● A central server algorithm
● A ring-based algorithm
● An algorithm using multicast and logic clocks (Ricart and Agarwal)
● Maekawa's voting algorithm

A central server algorithm



- The simplest way to achieve mutual exclusion is to employ a server that grants permission to enter the critical section.

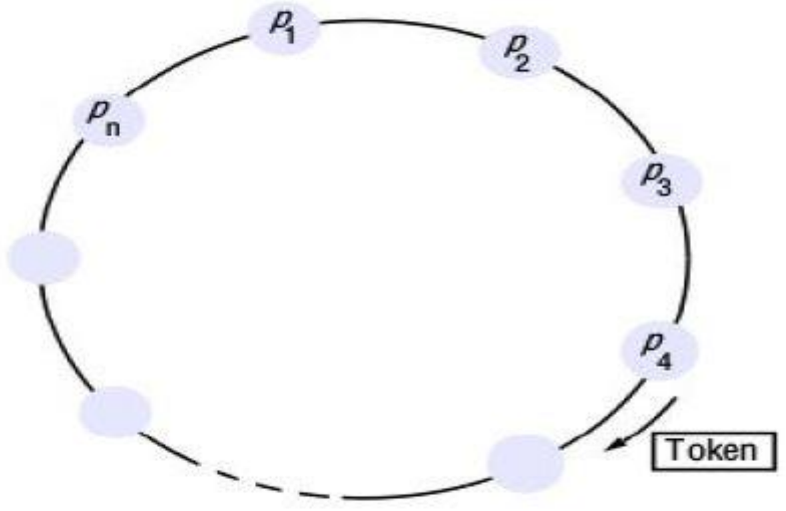
- To enter a critical section, a process sends a request message to he server and awaits a reply from it.
- Conceptually, the reply constitutes a token signifying permission to enter the critical section.
- If no other process has the token at the time of the request, then the server replies immediately, granting the token. If the token is currently held by another process, then the server does not reply, but queues the request. When a process exits the critical section, it sends a message to the server, giving it back the token.

Four process p1, p2, p3, p4

● p1- no need to enter critical section (CS)
● p4 & p2 send request the token to enter CS
● But p3 is accessing CS. So p4 & p2 enter the queue.
● P3 released its token
● p4 get the permission to enter CS
● After p4 released , permission grant to p2

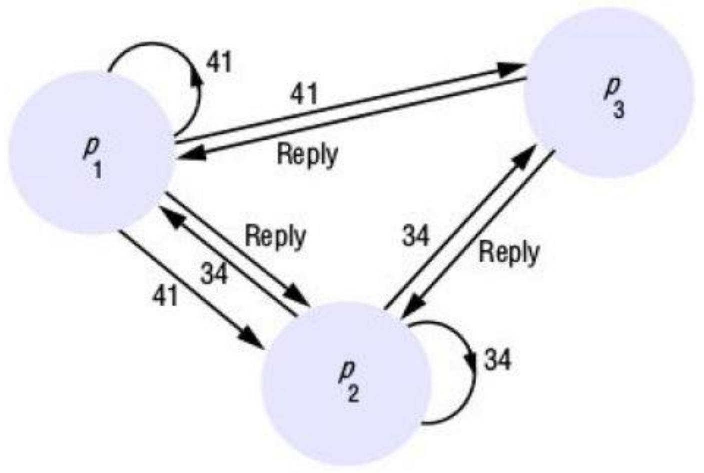
A ring-based algorithm

- One of the simplest ways to arrange mutual exclusion between the N processes without requiring an additional process is to arrange them in a logical ring.
- This requires only that each process p i has a communication channel to the next process in the ring, p (i + 1)mod N .
- The idea is that exclusion is conferred by obtaining a token in the form of a message passed from process to process in a single direction- clockwise.
- If a process does not require to enter the critical section when it receives the token, then it immediately forwards the token to its neighbour.
- A process that requires the token waits until it receives it, but retains it. To exit the critical section, the process sends the token on to its neighbour.



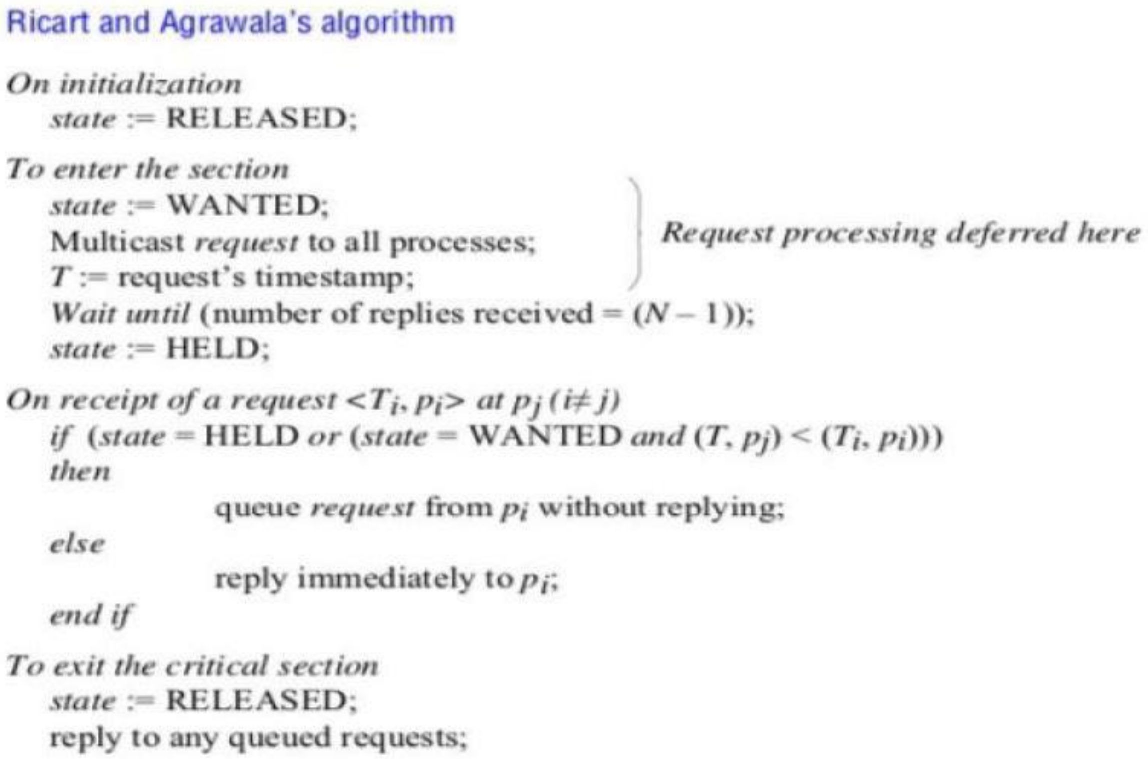
An algorithm using multicast and logic clocks (Ricart and Agarwal)

- Ricart and Agrawala [1981] developed an algorithm to implement mutual exclusion between N peer processes that is based upon multicast.
- The basic idea is that processes that require entry to a critical section multicast a request message, and can enter it only when all the other processes have replied to this message.
- The conditions under which a process replies to a request are designed to ensure that conditions ME1–ME3 are met.



To illustrate the algorithm, consider a situation involving three processes, p 1 , p 2 and p 3.

● p 3 is not interested in entering the critical section
● p 1 and p 2 request entry concurrently. Timestamp of p 1 ’s request is 41, and that of p 2 is 34
● p 3 receives their requests, it replies immediately
● p 2 receives p 1 ’s request, it finds that its own request has the lower timestamp and so does not reply
● p 1 finds that p 2 ’s request has a lower timestamp than that of its own request and so replies immediately.
● On receiving this second reply, p 2 can enter the critical section. When p 2 exits the critical section, it will reply to p 1 ’s request and so grant it entry.

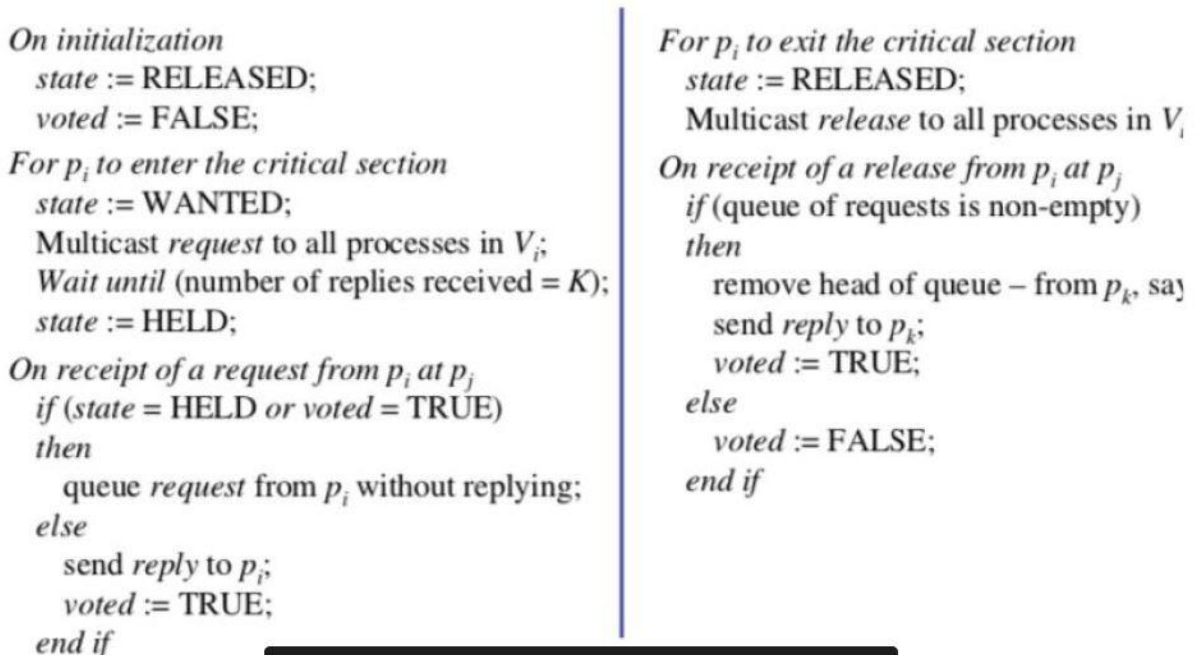


Ricart and Agrawala's algorithm

```
On initialization
    state := RELEASED;

To enter the section
    state := WANTED;                          }
    Multicast request to all processes;       } Request processing deferred here
    T := request's timestamp;                 }
    Wait until (number of replies received = (N – 1));
    state := HELD;

On receipt of a request <T_i, p_i> at p_j (i ≠ j)
    if (state = HELD or (state = WANTED and (T, p_j) < (T_i, p_i)))
    then
                queue request from p_i without replying;
    else
                reply immediately to p_i;
    end if

To exit the critical section
    state := RELEASED;
    reply to any queued requests;
```

Maekawa’s voting algorithm

- Maekawa observed that in order for a process to enter a critical section, it is not necessary for all of its peers to grant it access.
- Processes need only obtain permission to enter from subsets of their peers. A subset contain { p1, p2, p3, p4 }, if p1 wants to enter CS it need only permission from p2, p3 and p4. ie. A process pi send request msg to all other k-1 member of vi.
- This algorithm can tolerate some process crash failure. If a crashed process is not in a voting set then its failure will not affect the other process.



```
On initialization
    state := RELEASED;
    voted := FALSE;
For $p_i$ to enter the critical section
    state := WANTED;
    Multicast request to all processes in $V_i$;
    Wait until (number of replies received = $K$);
    state := HELD;
On receipt of a request from $p_i$ at $p_j$
    if (state = HELD or voted = TRUE)
    then
        queue request from $p_i$ without replying;
    else
        send reply to $p_i$;
        voted := TRUE;
    end if

For $p_i$ to exit the critical section
    state := RELEASED;
    Multicast release to all processes in $V$
On receipt of a release from $p_i$ at $p_j$
    if (queue of requests is non-empty)
    then
        remove head of queue – from $p_k$, sa
        send reply to $p_k$;
        voted := TRUE;
    else
        voted := FALSE;
    end if
```

- ➢ protocol

  – to obtain entry to critical section, pi sends request messages to all K-1 members of voting set Vi

  – cannot enter until K-1 replies received

  – when leaving critical section, send release to all members of Vi

  – when receiving request

    if state = HELD or already replied (voted) since last request * then queue request else immediately send reply

– when receiving release remove request at head of queue and send reply

- deadlocks are possible
  - – consider three processes with
    V1 = {p1, p2}, V2 = {p2, p3}, V3 = {p3, p1}
  - – possible to construct cyclic wait graph
    - p1 replies to p2, but queues request from p3
    - p2 replies to p3, but queues request from p1
    - p3 replies to p1, but queues request from p2

- algorithm can be modified to ensure absence of deadlocks
  - – use of logical clocks
  - – processes queue requests in happened-before order
  - – means that ME3 is also satisfied

ELECTIONS

- An algorithm for choosing a unique process to play a particular role (coordinator) is called an election algorithm. An election algorithm is needed for this choice. It is essential that all the processes agree on the choice.
- Afterwards, if the process that plays the role of server wishes to retire then another election is required to choose a replacement. We say that a process calls the election if it takes an action that initiates a particular run of the election algorithm.
- At any point in time, a process p i is either a participant – meaning that it is engaged in some run of the election algorithm – or a non-participant – meaning that it is not currently engaged in any election.
- Each process p i (i = 1 # 2 # } # N) has a variable elected i , which will contain the identifier of the elected process.
- When the process first becomes a participant in an election it sets this variable to the special value ‘ A ’ to denote that it is not yet defined. Our requirements are that, during any particular run of the algorithm:

E1: (safety) A participant process p i has elected i = A or elected i = P, where P is chosen as the non-crashed process at the end of the run with the largest identifier.

E2: (liveness) All processes p i participate and eventually either set elected i z A – or crash.

Two algorithms,

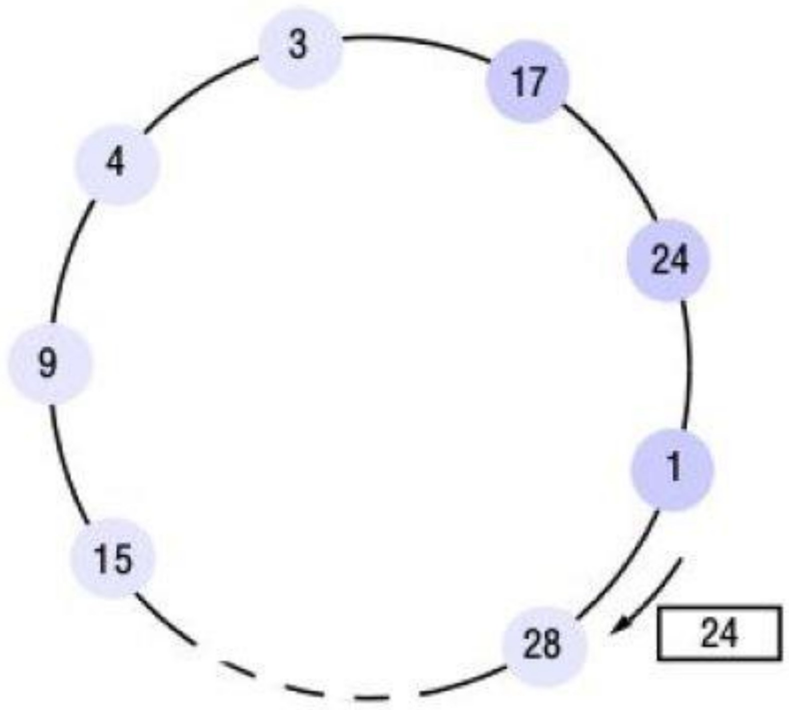
● **A ring-based election algorithm**

● **Bully algorithm**

A ring-based election algorithm

- The algorithm of Chang and Roberts is suitable for a collection of processes arranged in a logical ring.
- Each process p i has a communication channel to the next process in the ring, p (i + 1) mod N , and all messages are sent clockwise around the ring.
- The goal of this algorithm is to elect a single process called the coordinator, which is the process with the largest identifier.
- Initially, every process is marked as a non-participant in an election. Any process can begin an election.
- It proceeds by marking itself as a participant, placing its identifier in an election message and sending it to its clockwise neighbour.
- When a process receives an election message, it compares the identifier in the message with its own.
- If the arrived identifier is greater, then it forwards the message to its neighbour.
- If the arrived identifier is smaller and the receiver is not a participant, then it substitutes its own identifier in the message and forwards it; but it does not forward the message if it is already a participant.
- On forwarding an election message in any case, the process marks itself as a participant.
- If, however, the received identifier is that of the receiver itself, then this process's identifier must be the greatest, and it becomes the coordinator.

- The coordinator marks itself as a non-participant once more and sends an elected message to its neighbour, announcing its election and enclosing its identity.



i. The election was started by process 17. Process forward to neighbour with greatest identifier.
ii. The election message currently contains 24, and forwards
iii. But process 28 will replace this with its identifier when the message reaches it.
iv. The received identifier is that of the receiver itself, then this process's identifier must be the greatest, and it becomes the coordinator and sends an elected message to its neighbour,

The bully algorithm

- There are three types of message in this algorithm: an election message is sent to announce an election; an answer message is sent in response to an election message and a coordinator message is sent to announce the identity of the elected process.

- The process that knows it has the highest identifier can elect itself as the coordinator simply by sending a coordinator message to all processes with lower identifiers.
- On the other hand, a process with a lower identifier can begin an election by sending an election message to those processes that have a higher identifier and awaiting answer messages in response.
- If none arrives within time T, the process considers itself the coordinator and sends a coordinator message to all processes with lower identifiers announcing this.
- Otherwise, the process waits a further period Tc for a coordinator message to arrive from the new coordinator.If none arrives, it begins another election.
- If a process p i receives a coordinator message, it sets its variable elected i to the identifier of the coordinator contained within it and treats that process as the coordinator.
- If a process receives an election message, it sends back an answer message and begins another election – unless it has begun one already.

  - When a process, P, notices that the coordinator is no longer responding to requests, it initiates an election.

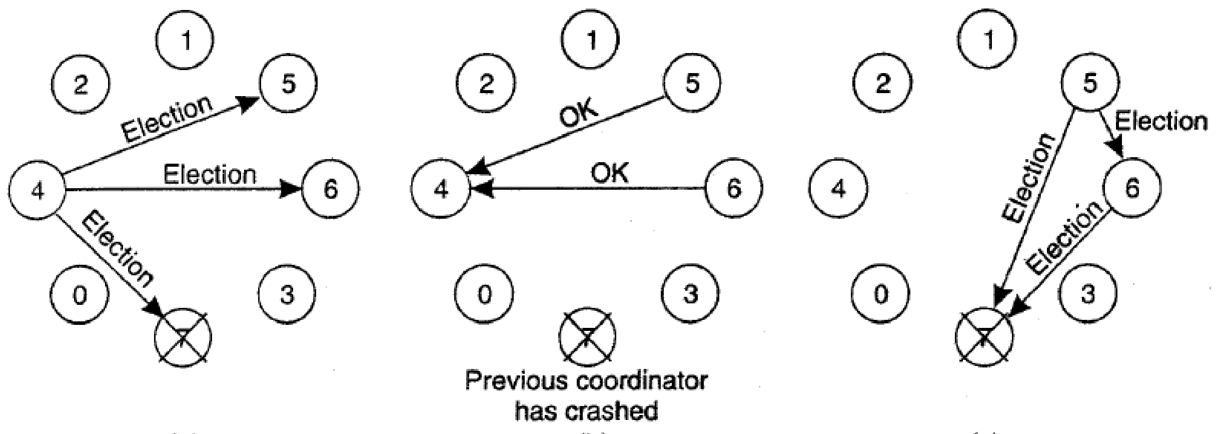
    - P sends an ELECTION message to all processes with higher no.
    - If no one responds, P wins the election and becomes a coordinator.
    - If one of the higher-ups answers, it takes over. P' s job is done.

  - When a process gets an ELECTION message from one of its lower-numbered colleagues:

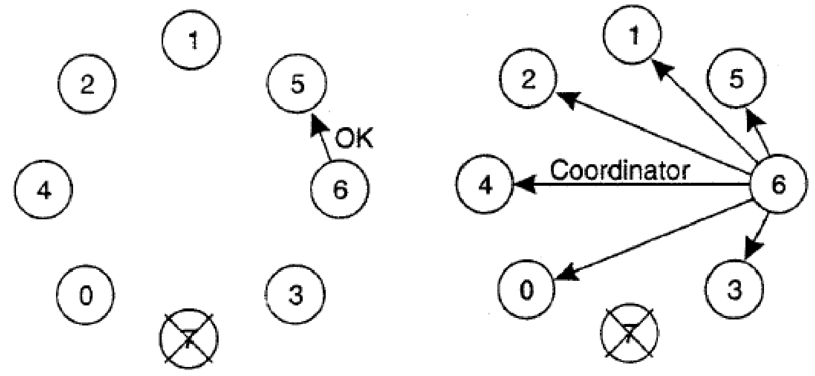
    - Receiver sends an OK message back to the sender to indicate that he is alive and will take over.
    - Receiver holds an election, unless it is already holding one.
    - Eventually, all processes give up but one, and that one is the new coordinator.

- The new coordinator announces its victory by sending all processes a message telling them that starting immediately it is the new coordinator.

➢ If a process that was previously down comes back:
  - It holds an election.
  - If it happens to be the highest process currently running, it will win the election and take over the coordinator' s job.
  - Biggest guy" always wins and hence the name " bully" algorithm.



(a) (b) (c)



CONTENT BEYOND SYLLABUS

DISTRIBUTED COMPUTING VS CLOUD COMPUTING

Cloud computing and distributed computing are two different systems but the fact that they both use the same concept means the two often leave people a little confused. To understand the two, you must first understand the underlying concept. It is simply the use of large-scale computer networks.

Distributed computing is the use of distributed systems to solve single large problems by distributing tasks to single computers in the distributing systems. On the other hand, cloud computing is the use of network hosted servers to do several tasks like storage, process and management of data. Here we will give an in-depth analysis of the two.

Cloud computing

Cloud computing has taken over the IT industry in the recent past. This is due to the fact that it is cheaper and easier to get services from the cloud. The cloud enables its users to choose how they will get and deliver IT services. Cloud computing means you can store and access data from the internet rather than the traditional computer hard disk storage.

This means you can access the data you have stored in the cloud anywhere anytime. The cloud will help you access the storage, servers, databases and multiple application services all in one place, the internet.

Benefits of cloud computing

The benefits of cloud computing are simply endless. Here we will list just a few:

Cost effective

The cloud helps you pay for the services you only require. Unlike building servers and databases, which are extremely expensive to build and maintain, the cloud helps you cut down that cost since you will pay for only what you are using.

Economies of Scale

By using the cloud, you will gain immensely from benefits of economies of scale. Simply put, you will get more value for money when using cloud rather than going solo.

Access to the global market

When using the cloud, you will have the chance of going global with a few clicks. You can reach the global audience without spending lots of cash and that’s not all, your customers will get superior services thanks to the cloud

Distributed computing

Distributed computing can simply be defined as sharing of tasks by different computers which may be in different parts of the globe. The distributed system which is used here must be in networked computers so that communication and coordination of the tasks is handled smoothly.

The main goal of distributed computing is to connect the users with the resources thereby maximizing the performance in a cost-effective way. It is also structured in a way that incase one of the components fails, the system goes on and the desired results are reached.

Benefits of distributed computing

There are many benefits of distributed computing. Below are just some of them;

Flexibility

One of the best thing about distributed computing is that it is highly flexible. Tasks can be completed using computers in different geographical areas.

Reliability

A single server can be rocked by glitches which can lead to complete systems malfunctions but with distributed computing, that is a thing of the past. With distributed computing, a single glitch cannot result to complete system failures.

Improved performance

Single computers can only perform to their best ability but with distributed computing, you get the best from across the whole system.

Both cloud computing use the same concept but individually they are two distinct things. As a business you can use both to improve your business and in return yield higher profits. Some of the examples of distributed computing are Facebook, World Wide Web and ATM. Examples of cloud computing are YouTube, Google Docs and Picasa.

SOFTWARE CONCEPT IN DISTRIBUTED COMPUTING

The software of the distributed system is nothing but selection of different operating system platforms.

The operating system is the interaction between user and the hardware.

There are three largely used operating system types:

a) Distributed operating system

b) Network operating system

c) Middleware operating system

Distributed operating system:

It is different from multiprocessor and multicomputer hardware.

Multiprocessor- uses different system services to manage resources connected in a system and use system calls to communicate with the processor.

Multicomputer- the distributed Operating system uses a separate uniprocessor OS on each computer for communicating between different computers.

In distributed OS, a common set of services is shared among multiple processors in such a way that they are meant to execute a distributed application effectively and also provide services to separate independent computers connected in a network as shown in fig below

It communicates with all the computer using message passing interface(MPI).

It follows the tightly coupled architecture pattern.

It uses Data structure like queue to manages the messages and avoid message loss between sender and receiver computer.

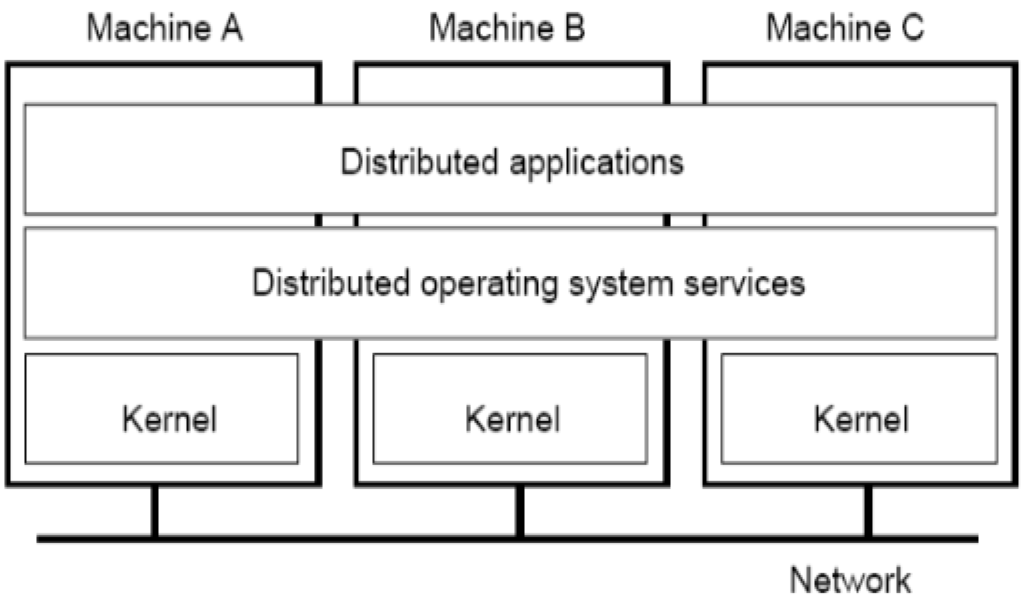
Eg Automated banking system, railway reservation system etc.

Disadvantages:

It has a problem of scalability as it supports only limited number of independent computers with shared resources.

There is need to define message passing semantics prior to the execution of messages.

Fig.



General structure of a multicomputer operating system

Network operating system:

It is specifically designed for hetrogeneous multicomputer system, where multiple hardware and network platforms are supported.

It has multiple operating system running on different hardware platforms connected in network.

It provides to each computer connected in network.

It follows the loosely coupled architecture pattern which allow user to use services provided by the local machine itself as shown in fig below.

Eg Remote login where user workstation is used to log in to the remoter server and execute remote commands over the network.

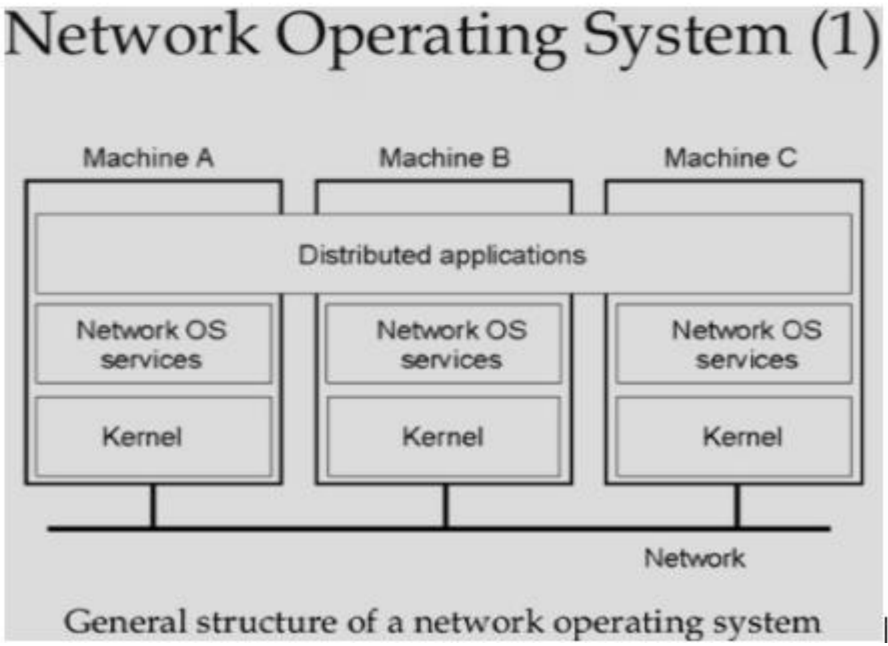
Eg Centralized file storage system.

Advantage:

It has scalability feature, where large number of resources and users are supported.

Disadvantage:

It fails to provide a single coherent view.



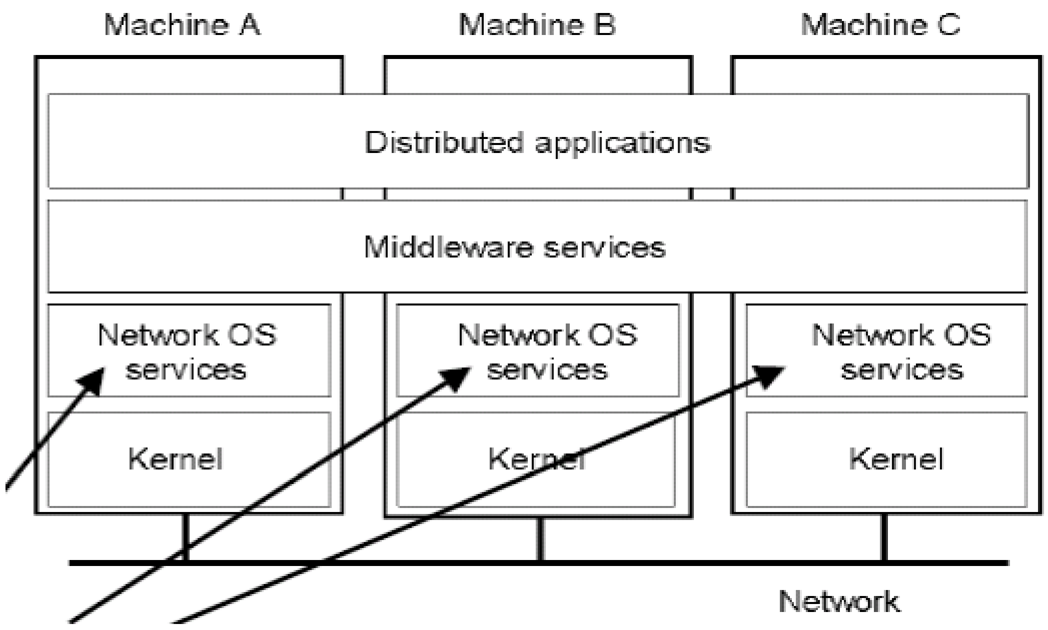
Middle ware operating system:

As distributed operating system has lack of scalability and network operating system fails to provide a single coherent view, therefore a new layer is formed between the distributed and network operating system is called the middleware operating system.

It has a common set of services is provided for the local applications and independent set of services for the remote applications.

It support heterogeneity that is it supports multiple languages and operating system where user gets freedom to write the application using the any of the supported language under any platform.

It provide the services such as locating the objects or interfaces by their names, finding the location of objects, maintaining the quality of services, handling the protocol information, synchronization, concurrency and security of the objects etc.



Fig(a) Middleware operating system