1

INTRODUCTION TO DISTRIBUTED SYSTEMS

1.1 WHAT IS DISTRIBUTED SYSTEM AND DISTRIBUTED COMPUTING?

The process of computation was started from working on a single processor. This uni-processor computing can be termed as **centralized computing**. As the demand for the increased processing capability grew high, multiprocessor systems came to existence. The advent of multiprocessor systems, led to the development of **distributed systems** with high degree of scalability and resource sharing. The modern day parallel computing is a subset of distributed computing.

A distributed system is a collection of independent computers, interconnected via a network, capable of collaborating on a task. Distributed computing is computing performed in a distributed system.

Distributed computing is widely used due to advancements in machines and faster and cheaper networks. In distributed systems, the entire network will be viewed as a computer. The multiple systems connected to the network will appear as a single system to the user. Thus the distributed systems hide the complexity of the underlying architecture to the user.

The definition of distributed systems deals with two aspects that:

- **Deals with hardware:** The machines linked in a distributed system are autonomous.
- **Deals with software:** A distributed system gives an impression to the users that they are dealing with a single system.

Features of Distributed Systems:

- Communication is hidden from users
- Applications interact in uniform and consistent way

- High degree of scalability
- A distributed system is functionally equivalent to the systems of which it is composed.
- Resource sharing is possible in distributed systems.
- Distributed systems act as fault tolerant systems
- Enhanced performance

Issues in distributed systems

- Concurrency
- Distributed system function in a heterogeneous environment. So adaptability is a major issue.
- Latency
- Memory considerations: The distributed systems work on both local and shared memory.
- Synchronization issues
- Applications must need to adapt gracefully without affecting other parts of the systems in case of failures.
- Since they are widespread, security is a major issue.
- Limits imposed on scalability
- They are less transparent.
- Knowledge about the dynamic network topology is a must.

QOS parameters

The distributed systems must offer the following QOS:

- Performance
- Reliability
- Availability
- Security

Differences between centralized and distributed systems

Centralized Systems	Distributed Systems
In Centralized Systems, several jobs are done on a particular central processing unit(CPU)	In Distributed Systems, jobs are distributed among several processors. The Processor are interconnected by a computer network
They have shared memory and shared variables.	They have no global state (i.e.) no shared memory and no shared variables.
Clocking is present.	No global clock.

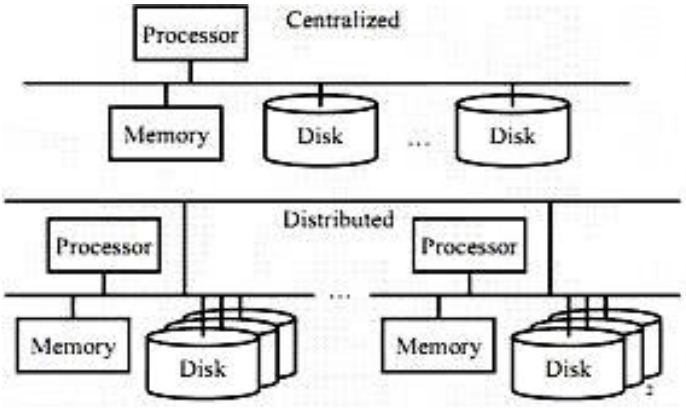


Fig 1.1: Distributed and Centralized systems

1.2 SOME EXAMPLES OF DISTRIBUTED SYSTEMS

Today's internet exists on the distributed systems. There are numerous applications of distributed systems. Some of them are discussed below:

Web Search

- ✓ The task of a web search engine is to index the entire contents of the World Wide Web.
- ✓ Distributed search is a search engine model in which the tasks of Web crawling, indexing and query processing are distributed among multiple computers and networks.
- ✓ The search engines were supported by a single supercomputer . But in recent years, they have moved to a distributed model.
- ✓ Google search relies upon thousands of computers crawling the Web from multiple locations all over the world.

- ✓ In Google's distributed search system, each computer involved in indexing crawls and reviews a portion of the Web, taking a URL and following every link available from it.
- ✓ The computer gathers the crawled results from the URLs and sends that information back to a centralized server in compressed format.
- ✓ The centralized server then coordinates that information in a database, along with information from other computers involved in indexing.
- ✓ When a user types a query into the search field, Google's domain name server (DNS) software relays the query to the most logical cluster of computers, based on factors such as its proximity to the user or how busy it is.
- ✓ At the recipient cluster, the Web server software distributes the query to hundreds or thousands of computers to search simultaneously.
- ✓ Hundreds of computers scan the database index to find all relevant records.
- ✓ The index server compiles the results, the document server pulls together the titles and summaries and the page builder creates the search result pages.
- ✓ The following features of Google search makes it act as distributed system:
  - ➢ The physical infrastructure with very large numbers of networked computers .
  - ➢ Highly distributed file system that supports very large files.
  - ➢ Availability of structured distributed storage system for fast access to data.
  - ➢ Distributed locking and agreement.
  - ➢ Works on a programming model that supports the management of large parallel and distributed computations.

Massively multiplayer online games (MMOGs)

- ✓ These games simulate real-life as much as possible.
- ✓ As such it is necessary to constantly evolve the game world using a set of laws.
- ✓ Theselaws are a complex set of rules that the game engine applies with every clock tick.
- ✓ The virtual world consists not only of human players but also of all game elements that are not living objects.
- ✓ These elements are immutable and include the area terrain, trees, mountains, rivers, etc.

- ✓ MMOGs must be able to handle a very large number of simultaneous users.
- ✓ As the information transferred between the players and the game server is large, the bandwidth required to support a huge number of players if enormous.
- ✓ Very large virtual worlds require huge computational power to simulate the existence of life (AI Algorithms).
- ✓ No single processor machine can handle the computational load required.
- ✓ These gaming applications work on both client server and distributed architectures.
- ✓ In distributed architectures, the large virtual world of the game is split into different smaller areas and each area is to be handled by a separate physical machine (server).
- ✓ Therefore both the bandwidth and computational load is spread out on many machines, thus making the application distributed.

Financial Trading

- ✓ The financial trading is now moving to distributed systems.
- ✓ These systems require frequent modifications, in response to the communication.
- ✓ Processing the events in distributed systems, demands reliability.
- ✓ So distributed event based systems are used.
- ✓ A series of event feeds are given by the financial institution.
- ✓ The events may be in different formats, employ different technologies etc.
- ✓ So proper adaptors are a must.
- ✓ Pattern detection is also very import aspect in financial trading.
- ✓ The **Complex Event Processing (CEP)** is an automatized way of composing event together into logical, temporal or spatial patterns.

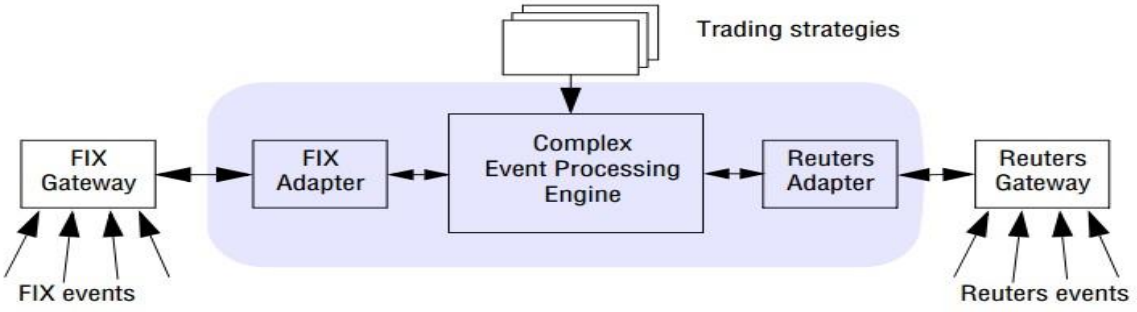


Fig 1.2: Financial Trading System

1.3 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.

1.3.1Pervasive networking and the modern Internet

Pervasive computing devices are completely connected and constantly available.

- ✓ The products that are connected to the pervasive network are easily available.
- ✓ The main goal of pervasive computing is to create an environment where the connectivity of devices is embedded in such a way that the connectivity is unobtrusive and always available.
- ✓ Internet can be seen as a large distributed system.

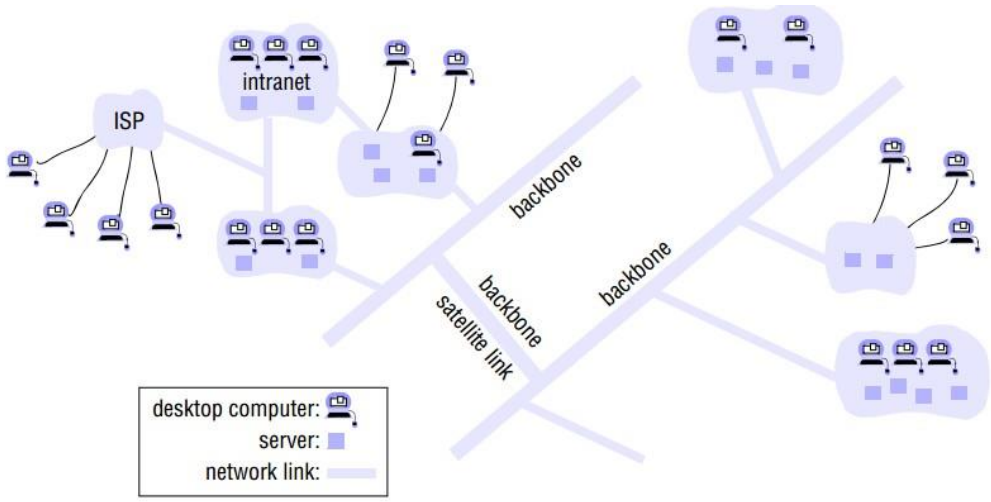


Fig 1.3: Internet

- ✓ Internet is actually a collection of numerous sub networks operated by companies and other organizations and typically protected by firewalls.
- ✓ The role of a firewall is to protect an intranet by preventing unauthorized messages from leaving or entering.

- ✓ Internet Service Providers (ISPs) are companies that provide broadband links and other types of connection to individual users and small organizations, enabling them to access services anywhere in the Internet as well as providing local services such as email and web hosting.

- ✓ The intranets are linked together by backbones network link with a high transmission capacity, employing satellite connections, fibre optic cables and other high-bandwidth circuits.

1.3.2 Mobile and ubiquitous computing

- ✓ Internet is now build into numerous small devices from laptops to watches.

- ✓ These devices must have high degree of portability. Mobile computing supports this.

- ✓ Mobile computing is the performances of computing tasks while the user dynamically changing his geographic location.

- ✓ Ubiquitous computing (small computing) means that all small computing devices will eventually become so pervasive in everyday objects.

Differences between ubiquitous computing and mobile computing

Ubiquitous Computing	Mobile Computing
They could connect tens/hundreds of computing devices in every room/person, becoming "invisible" and part of the environment – WANs, LANs, PANs – networking in small spaces	They could connect a few devices for every person, small enough to carry around – devices connected to cellular networks or WLANs
They could connect even the non- mobile devices and offer various forms of communication.	They are actually a subset of ubiquitous computing.
They could support all form of devices that are connected to the internet from laptops to watches.	They support only conventional, discrete computers and devices.

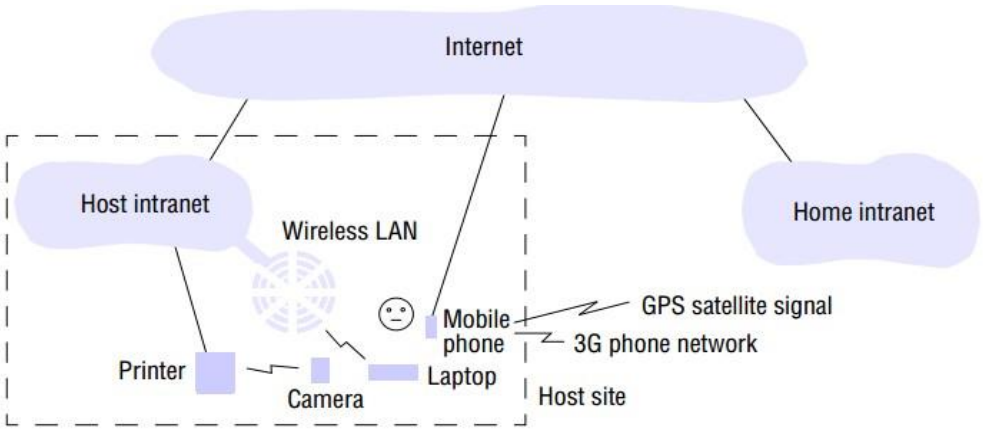


Fig 1.4: Mobile devices in an environment

- ✓ The user in the above given environment has access to three forms of wireless connection:
  - ∗ The laptop connecting to the host's wireless LAN.
  - ∗ Mobile (cellular) telephone connected to the Internet. The phone gives access to the Web and other Internet services.
  - ∗ Digital camera that communicates over a personal area wireless network.
- ✓ This scenario demands associations between devices are routinely created and destroyed called as **spontaneous interoperation.** This interoperation must be fast and convenient.

1.3.3 Distributed multimedia systems

- ✓ A distributed system supports the storage, transmission and presentation of discrete media types.
- ✓ A distributed multimedia system should be able to perform the same functions for continuous media types such as audio and video.
- ✓ It should be able to store and locate audio or video files, to transmit them across thenetwork to support the presentation of the media types to the user and optionally also to share the media types across a group of users.
- ✓ The processing of such media files includes handling of temporal dimensions and integrity of media.
- ✓ The distributed multimedia computing allows a wide range of new multimedia services and applications to be provided on the desktop.

- ✓ This includes 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.

- ✓ Webcasting demands the following changes in the infrastructure:
  - ➢ Supportfor an range of encoding and encryption formats.
  - ➢ Support for range of mechanisms to ensure that the desired quality of service canbe met.
  - ➢ Adaptability to associated resource management strategies.
  - ➢ Providing adaptation strategies to deal with the situation where QOS is difficlut to achieve.

1.3.4 Distributed computing as a utility

- ✓ Distributed systems are seen as a utility like water and electricity.
- ✓ The resources are provided by appropriate service suppliers and effectively rented by the end user.
- ✓ The services may be physical or logical services.
- ✓ Physical resources such as storage and processing can be made available to networked computers through data centres.
- ✓ Operating system virtualization is a key enabling technology that users may actually be provided with services by a virtual rather than a physical node.
- ✓ Software services rental, services such as email and distributed calendars.
- ✓ Cloud computing is the child of resource sharing.

A cloud is defined as a set of Internet-based application, storage and computing services sufficient to support most users' needs, thus enabling them to largely or totally dispense with local data storage and application softwarewith local data storage and application software.

- ✓ Clouds are generally implemented on cluster computers.
- ✓ A cluster computer is a set of interconnected computers that cooperate closely to provide a single, integrated high performance computing capability.

1.4 RESOURCE SHARING

- ❖ An important goal of a distributed system is to effectively utilize the collective resources of the system, namely, the memory and the processors of the individual nodes.
- ❖ Users think in terms of shared resources such as a search engine without regard for the server or servers that provide these.
- ❖ A search engine on the Web provides a facility to users throughout the world, users who need never come into contact with one another directly.
- ❖ In **computer-supported cooperative working (CSCW),** a group of users who cooperate directly share resources such as documents in a small, closed group.
- ❖ The pattern of sharing and the geographic distribution of particular users determine what mechanisms the system must supply to coordinate users' actions.
- ❖ The **service** manages a collection of related resources and presents their functionality to users and applications.
- ❖ Resources in a distributed system are physically encapsulated within computers and can only be accessed from other computers by means of communication.
- ❖ Sharing is done by a program that offers a communication interface enabling the resource to be accessed and updated reliably and consistently.
- ❖ In a client server paradigm, server refers to a running program on a networked computer that accepts requests from programs running on other computers to perform a service and responds appropriately.
- ❖ The requesting processes are referred to as clients.
- ❖ A complete interaction between a client and a server, from the point when the client sends its request to when it receives the server's response, is called a **remote invocation.**

1.5 CHALLENGES IN DISTRIBUTED SYSTEMS

a) Heterogeneity

Heterogeneity means the diversity of the distributed systems in terms of hardware, software, platform, etc. Modern distributed systems will likely to be operating with different:

- ➢ Hardware devices: computers, tablets, mobile phones, embedded devices, etc.

- Operating System: Ms Windows, Linux, Mac, Unix, etc.
- Network: Local network, the Internet, wireless network, satellite links, etc.
- Programming languages: Java, C/C++, Python, PHP, etc.
- Different roles of software developers, designers, system managers
- Middleware: 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. Eg: CORBA, RMI.
- Heterogeneity in mobile code: Mobile code is used to refer to program code that can be transferred from one computer to another and run at the destination.

  Eg: Java applets.

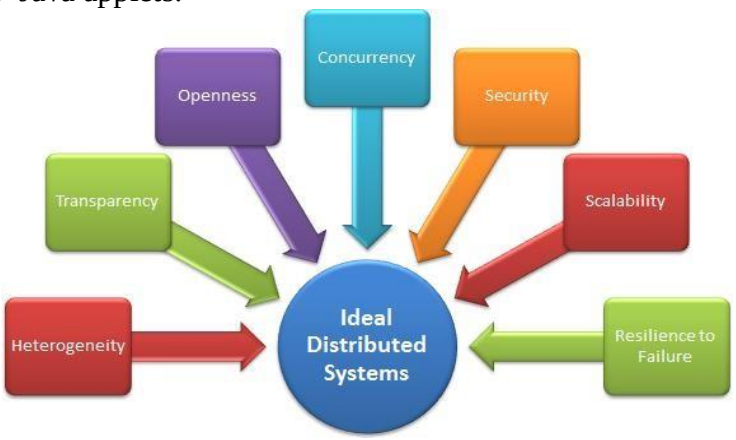


Fig 1.5: Challenges in Distributed systems

b) Transparancy

Distributed systems designers must hide the complexity of the systems. Adding abstraction layer is particularly useful in distributed systems. While users hit search in google.com, they never notice that their query goes through a complex process before google shows them a result. Some terms of transparency in distributed systems are:

Transparency	**Description**
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located

Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource may be copied in several places
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or a disk

The access and location transparency is collectively referred as **network transparency.**

c) Openness

If the well-defined interfaces for a system are published, it is easier for developers to add new features or replace sub-systems in the future. Example: Twitter and Facebook have API that allows developers to develop their software. The following are key points in openness:

* key interfaces are published.
* 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.

d) Concurrency

Distributed Systems usually is multi-users environment. In order to maximize concurrency, resource handling components should anticipate as they will be accessed by competing users. Concurrency prevents the system to become unstable when users compete to view or update data.

e) Security

Every system must consider strong security measurement. Distributed Systems somehow deals with sensitive information; so secure mechanism must be in place. The foolowing attacks are more common in distributed systems:

* **Denial of service attacks**: When the requested service is not available at the time of request it is Denial of Service (DOS) attack. This attack is done by bombarding the service with a large number of useless requests that the serious users are unable to use it.

- **Security of mobile code:** Mobile code needs to be handled with care since they are transmitted in an open environment.

f) Scalability

Distributed systems must be scalable as the number of user increases.

> ***A system is said to be scalable if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity.***

Scalability has 3 dimensions:

- **Size:** Number of users and resources to be processed. Problem associated with this is overloading.
- **Geography:** Distance between users and resources. Problem associated is communication reliability
- **Administration**: As the size of distributed systems increases, many of the system needs to be controlled. Problem associated is administrative mess

Scalability often conflicts with small system performance. Claim of scalability in such system is often abused. The following are the scalability challenges:

- Controlling the cost of physical resources
- Controlling the performance loss
- Preventing software resources running out
- Avoiding performance bottlenecks

g) Resilience to Failure (Fault tolerance)

Distributed Systems involves a lot of collaborating components (hardware, software, communication). So there is a huge possibility of partial or total failure. The failures are handled in series of steps:

- **Detecting failures:** Some failures like checksum can be detected.
- **Masking failures:** Some failures that have been detected can be hidden or made less severe. Examples of hiding failures include retransmission of messages and maintaining a redundant copy of same data.
- **Tolerating failures:** All the failures cannot be handled. Some failures must be accepted by the user. Example of this is waiting for a video file to be streamed in.

- **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.
- **Redundancy**: Services can be made to tolerate failures by the use of redundant components. Examples for this includes: maintenance of two different paths between same source and destination.
- Availability is also a major concern in the fault tolerance. The **availability**of a system is a measure of the proportion of time that it is available for use. It is a useful performance metric.

h) Quality of Service:

The distributed systems must confirm the following non functional requirements:

- **Reliability:**A reliable distributed system is designed to be as fault tolerant as possible. Reliability is the quality of a measurement indicating the degree to which the measure is consistent.
- **Security:** Security is the degree of resistance to, or protection from, harm. It applies to any vulnerable and valuable asset, such as a person, dwelling, community, nation, or organization. Distributed systems spread across wide geographic locations. So security is a major concern.
- **Adaptability:**The frequent changing of configurations and resource availability demands the distributed system to de highly adaptable.

1.6 WORLD WIDE WEB (WWW)

1.6.1 History and Development of WWW

- ✓ The World Wide Web (WWW) can be viewed as a huge distributed system consisting of millions of clients and servers for accessing linked documents.
- ✓ Servers maintain collections of documents, while clients provide users an easy to-use interface for presenting and accessing those documents.
- ✓ The Web started as a project at CERN, the European Particle Physics Laboratory in Geneva, to let its large and geographically dispersed group of researchers provide access to shared documents using a simple hypertext system.
- ✓ A document in a WWW could be anything that could be displayed on a user's computer terminal, such as personal notes, reports, figures, blueprints, drawings, and so on.
- ✓ By linking documents to each other, it became easy to integrate documents from different projects into a new document without the necessity for centralized changes.

- ✓ The Web gradually grew worldwide encompassing sites other than high energy physics, but popularity really increased when graphical user interfaces became available, notably Mosaic (Vetter et al., 1994).
- ✓ Mosaic provided an easy-to-use interface to present and access documents by merely clicking the mouse.
- ✓ A document was fetched from a server, transferred to a client, and presented on the screen.
- ✓ To a user, there was conceptually no difference between a document stored locally or in another part of the world. This is transparent distribution.
- ✓ Since 1994, Web developments are primarily initiated and controlled by the World Wide Web Consortium, which is a collaboration between CERN and M.I.T.
- ✓ This consortium is responsible for standardizing protocols, improving interoperability, and further enhancing the capabilities of the Web.
- ✓ The webpages are portable and open.

1.6.2 Components of WWW

- **HTML**
  - ✓ The Web documents are expressed by means of a special language called HyperText Markup Language (HTML).
  - ✓ HTML provides keywords to structure a document into different sections.
  - ✓ One of its most powerful features is the ability to express parts of a document in the form of a script.
  - ✓ When a document is parsed, it is internally stored as a rooted tree, called a **parse tree**, in which each node represents an element of that document.
  - ✓ Each node is required to implement a standard interface containing methods for accessing its content, returning references to parent and child nodes, and so on. This standard representation is also known as the **Document Object Model** or DOM or **dynamic HTML**.
  - ✓ An alternative language that also matches the DOM is XML (Extensible Markup Language).
  - ✓ XML is used only to structure a document; it contains no keywords.
  - ✓ XML can be used to define arbitrary structures. In other words, it provides the means to define different document types.

- **Uniform Resource Locators (URL)**

The pages of a website can usually be accessed from a simple Uniform Resource Locator (URL) otherwise called as web address.

- ✓ URL's are a way of identifying information on a server.
- ✓ A URL gives the protocol, the domain, the directory, and even the file.
- ✓ A URL consists of the following parts:
  - protocol (such as http:// or ftp://)
  - host name (the Web server's IP address or domain name)
  - directory (i.e. folder)
  - file name

There are two forms of URL as listed below:

- **Absolute URL:**

  Absolute URL is a complete address of a resource on the web. This completed address comprises of protocol used, server name, path name and file name.

  Example: http:// www.abc.com / xyz /index.htm.

  - http is the protocol.
  - abc.com is the server name.
  - index.htm is the file name.

  The protocol part tells the web browser how to handle the file. Other protocols also that can be used to create URL are: FTP, https, Gophe, mailto, news

- **Relative URL**

  Relative URL is a partial address of a webpage. Unlike absolute URL, the protocol and server part are omitted from relative URL. Relative URLs are used for internal links i.e. to create links to file that are part of same website as the WebPages on which you are placing the link.

  Example:

  To link an image on abc.com/xyz/internet_referemce_models, we can use the relative URL which can take the form like /internet_technologies/internet-osi_model.jpg.

- **Hyper Text Transfer Protocol (HTTP)**
  - ✓ HTTP is a communication protocol.
  - ✓ It defines mechanism for communication between browser and the web server. It is also called request and response protocol because the communication between browser and server takes place in request and response pairs.
  - ✓ It is a **stateless** protocol (i.e.) the history of the communication between server and client is not stored in any form.

HTTP Request

- ✓ HTTP request comprises of lines which contains: Request line, Header Fields and Message body.
- ✓ The first line i.e. the Request line specifies the request method i.e. Get or Post.
- ✓ The second line specifies the header which indicates the domain name of the server from where index.htm is retrieved.

HTTP Response

- ✓ Like HTTP request, HTTP response also has certain structure.
- ✓ HTTP response contains: Status line, Headers and Message body.

- **Publishing a resource:**
  - ✓ The methods for publishing resources on the Web are dependent upon the web server implementation.
  - ✓ The simplest method of publishing a resource on the Web is to place the corresponding file in a directory that the web server can access.
  - ✓ It is common for such concerns to be hidden from users when they generate content.
  - ✓ The database or file system on which the product pages are based is transparent.

- **Downloaded code:**
  - ✓ The designers of web services require some service-related code to run inside the browser, at the user's computer.
  - ✓ Javascript is an example for downloaded code.
  - ✓ A Javascript enhanced page can give the user immediate feedback on invalid entries, instead of forcing the user to check the values at the server, which would take much longer.

- ✓ AJAX (Asynchronous Javascript And XML) is used in cases where synchronization is not a major concern.
- ✓ Applet is also a form of downloaded code.
- ✓ It is an application which the browser automatically downloads and runs when it fetches a corresponding web page.

- **Web services**
  - ✓ Web services allow exchange of information between applications on the web.
  - ✓ Using web services, applications can easily interact with each other.
  - ✓ The web services are offered using concept of Utility Computing.

REVIEW QUESTIONS

PART-A

1. Define centralized computing.

The process of computation was started from working on a single processor. This uni-processor computing can be termed as centralized computing.

2. Define distributed systems.

A distributed system is a collection of independent computers, interconnected via a network, capable of collaborating on a task. Distributed computing is computing performed in a distributed system.

3. List the features of Distributed Systems.

- Communication is hidden from users
- Applications interact in uniform and consistent way
- High degree of scalability
- A distributed system is functionally equivalent to the systems of which it is composed.
- Resource sharing is possible in distributed systems.
- Distributed systems act as fault tolerant systems
- Enhanced performance

4. What are the issues in distributed systems?

- Concurrency
- Distributed system function in a heterogeneous environment. So adaptability is a major issue.
- Latency
- Memory considerations: The distributed systems work on both local and shared memory.
- Synchronization issues
- Applications must need to adapt gracefully without affecting other parts of the systems in case of failures.
- Since they are widespread, security is a major issue.

- Limits imposed on scalability
- They are less transparent.
- Knowledge about the dynamic network topology is a must.

5. Mention the QOS parameters of DS.

The distributed systems must offer the following QOS:

- Performance
- Reliability
- Availability
- Security

6. Give the differences between centralized and distributed systems

Centralized Systems	**Distributed Systems**
In Centralized Systems, several jobs are done on a particular central processing unit(CPU)	In Distributed Systems, jobs are distributed among several processors. The Processor are interconnected by a computer network
They have shared memory and shared variables.	They have no global state (i.e.) no shared memory and no shared variables.
Clocking is present.	No global clock.

7. What is distributed search?

Distributed search is a search engine model in which the tasks of Web crawling, indexing and query processing are distributed among multiple computers and networks.

8. List the features of Google DS.

The physical infrastructure with very large numbers of networked computers .

Highly distributed file system that supports very large files.

Availability of structured distributed storage system for fast access to data.

Distributed locking and agreement.

Works on a programming model that supports the management of large parallel and distributed computations.

9. What is MMOG?

These games simulate real-life as much as possible. As such it is necessary to constantly evolve the game world using a set of laws.These laws are a complex set of rules that the game engine applies with every clock tick

10. What is CEP?

The Complex Event Processing (CEP) is an automatized way of composing event together into logical, temporal or spatial patterns.

11. What is pervasive computing?

Pervasive computing devices are completely connected and constantly available.

12. Define ISP.

Internet Service Providers (ISPs) are companies that provide broadband links and other types of connection to individual users and small organizations, enabling them to access services anywhere in the Internet as well as providing local services such as email and web hosting.

13. Define Ubiquitous computing .

Ubiquitous computing (small computing) means that all small computing devices will eventually become so pervasive in everyday objects.

14. Give the differences between ubiquitous computing and mobile computing

Ubiquitous Computing	**Mobile Computing**
They could connect tens/hundreds of computing devices in every room/person, becoming "invisible" and part of the environment – WANs, LANs, PANs – networking in small spaces	They could connect a few devices for every person, small enough to carry around – devices connected to cellular networks or WLANs
They could connect even the non- mobile devices and offer various forms of communication.	They are actually a subset of ubiquitous computing.
They could support all form of devices that are connected to the internet from laptops to watches.	They support only conventional, discrete computers and devices.

15. Define webcasting.

Webcasting is the ability to broadcast continuous media, typically audio or video, over the Internet.

16. List the requirements of web casting.

Support for an range of encoding and encryption formats.

Support for range of mechanisms to ensure that the desired quality of service can be met.

Adaptability to associated resource management strategies.

Providing adaptation strategies to deal with the situation where QOS is difficlut to achieve.

17. Define cloud.

A cloud is defined as a set of Internet-based application, storage and computing services sufficient to support most users' needs, thus enabling them to largely or totally dispense with local data storage and application software with local data storage and application software.

18. What is CSCW?

In computer-supported cooperative working (CSCW), a group of users who cooperate directly share resources such as documents in a small, closed group.

19. Define remote invocation.

A complDEete interaction between a client and a server, from the point when the client sends its request to when it receives the server's response, is called a remote invocation.

20. List the challenges in DS.

- ↑ Heterogeneity
- ↑ Transparency
- ↑ Openness
- ↑ Concurrency
- ↑ Security
- ↑ Scalability
- ↑ Fault tolerance
- ↑ Providing QOS

21. What are the dimensions of scalability?

* Size: Number of users and resources to be processed. Problem associated with this is overloading.

* Geography: Distance between users and resources. Problem associated is communication reliability

* Administration: As the size of distributed systems increases, many of the system needs to be controlled. Problem associated is administrative mess

22. List the challenges in scalability.

The following are the scalability challenges:

- Controlling the cost of physical resources
- Controlling the performance loss
- Preventing software resources running out
- Avoiding performance bottlenecks

23. Give the features in fault tolerance.

Detecting failures: Some failures like checksum can be detected.

Masking failures: Some failures that have been detected can be hidden or made less severe. Examples of hiding failures include retransmission of messages and maintaining a redundant copy of same data.

Tolerating failures: All the failures cannot be handled. Some failures must be accepted by the user. Example of this is waiting for a video file to be streamed in.

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.

Redundancy: Services can be made to tolerate failures by the use of redundant components. Examples for this includes: maintenance of two different paths between same source and destination.

24. What is parse tree?

When a document is parsed, it is internally stored as a rooted tree, called a parse tree, in which each node represents an element of that document.

25. Define URL.

The pages of a website can usually be accessed from a simple Uniform Resource Locator (URL) otherwise called as web address.

PART – B

1. What are the issues in distributed systems?
2. Explain web search as a distributed system.
3. Brief about MMOGs.
4. Describe financial trading.
5. Explain the trends in distributed systems.
6. Elucidate the resource sharing nature of distributed system.
7. Brief about the challenges in distributed system.
8. Describe WWW.
9. Explain HTML.

2

COMMUNICATION IN DISTRIBUTED SYSTEM

2.1 COMMUNICATION IN DISTRIBUTED SYSTEMS

The most important difference between a distributed system and a uniprocessor system is the **interprocess communication.**

Interprocess communication (IPC) is a set of programming* interfaces *that allows a programmer to coordinate activities among different program* processes *that can run concurrently in an operating system.

- ✓ In a uniprocessor system, interprocess communication assumes the existence of shared memory whereas in a distributed system, there‟s no shared memory, so the entire nature of interprocess communication must be completely reframed from scratch.
- ✓ All communication in distributed system is based on **message passing**.
- ✓ The processes run on different machines and they exchange information through message passing.
- ✓ Successful distributed systems depend on communication models that hide or simplify message passing.

2.2 SYSTEM MODELS

System models describe common properties and design choices for distributed system in a single descriptive model. The system models of distributed systems are classified into the following types:

- **Physical models:** It is the most explicit way in which to describe a system in terms of hardware composition.
- **Architectural models**: They describe a system in terms of the computational and communication tasks performed by its computational elements.
- **Fundamental models**: They examine individual aspects of a distributed system. They are again classified based on some parameters as follows:
  - ✓ **Interaction models:** This deals with the structure and sequencing of the communication between the elements of the system
  - ✓ **Failure models:** This deals with the ways in which a system may fail to operate correctly
  - ✓ **Security models:** This deals with the security measures implemented in the system against attempts to interfere with its correct operation or to steal its data.

2.2.1 Physical Models

A physical model is a representation of the underlying hardware elements of a distributed system hides the details of the computer and networking technologies employed.

There are many physical models available from the primitive baseline model to the complex models that could handle cloud environments.

- **Baseline physical model:**
  - ✓ This is a primitive model that describes the hardware or software components located at networked computers.
  - ✓ The communication and coordination of their activities is done by passing messages.
  - ✓ This is a minimal physical model of a distributed system.
  - ✓ This model could be extended to a set of computer nodes interconnected by a computer network for the required passing of messages.
  - ✓ This model helps us to categorize the three generations of distributed systems.
- **Early distributed systems**:
  - ✓ The period is in the late 1970s and early 1980s.
  - ✓ This was developed after the emergence LAN.

- ✓ These systems could support 10 to 100 nodes interconnected by a LAN.
- ✓ It has very limited internet connectivity and can support a small range of services such as shared local printers and file servers.
- ✓ Individual systems were homogeneous, with poor quality of service.

➢ **Internet-scale distributed systems:**

- ✓ The period is from the 1990s after the growth of internet.
- ✓ The physical set up of distributed system is described as an extensible set of nodes interconnected by a network of networks (the Internet).
- ✓ They incorporate large numbers of nodes and provide distributed system services.
- ✓ The systems were heterogeneous which could adopt open standards.
- ✓ They had good QOS.

➢ **Contemporary distributed systems:**

- ✓ The nodes were desktop computers and therefore relatively static, discrete and independent.
- ✓ Mobile computing has led to physical models with movable nodes. Service discovery is of primary concern in these systems.
- ✓ The cloud computing and cluster architectures hassled to pools of nodes that collectively provide a given service. This resulted in enormous number of systems given the service.

➢ **Distributed systems of systems:**

The ultra large scale (ULS) distributed systems is 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 or tasks. Some important characteristics of these systems include their

- very large size
- global geographical distribution
- operational and managerial independence of their member systems.

The main function of these systems arises from the interoperability between their components.

Example: Flood monitoring systems.

Parameter	Early Systems	Internet Scale	ULS System
Scale	Small	Large	Ultra Large
Heterogeneity	Homogeneous	Platform independent software and technologies.	Supports many types of architectures
Openness	Systems are mostly closed	Many open standards are available	The already existing standards are not able to cope with complex systems.
QOS	Poor	Significant	QOS could still be improved

2.2.2 Architectural Models

The architecture abstracts the functions of the individual components of the distributed system.

This describes the components and their interrelationships to ensure the system is reliable, manageable, adaptable and cost-effective. The different models are evaluated based on the following criteria:

- architectural elements
- architectural patterns
- middleware platforms

2.2.2.1 Architectural elements

The following must be decided to build a distributed system:

- Communicating entities (objects, components and web services);
- Communication paradigms (inter process communication, remote invocation and indirect communication);
- Roles responsibilities and placement

Communicating entities

The components that are communicating and how those entities communicate together are dealt here. The following are some of the problem oriented entities:

- **Objects:** They are used to implement object oriented approaches in distributed systems. Objects are accessed through interfaces.
- **Components:** Components resemble objects and they offer problem-oriented abstractions for building distributed systems. They are also accessed through interfaces. The key difference between component and object is that a components holds all the assumptions specified to other components and their interfaces in the system. Components and objects are used to develop tightly coupled applications.
- **Web services:** Web services are closely related to objects and components. Web services are integrated into the World Wide Web. They are partially defined by the web-based technologies they adopt. Web services are generally viewed as complete services that can be combined to achieve value-added services across organizational boundaries.

Object	**Components**	**Web services**
Object oriented solution	Problem oriented solution	Problem oriented solution
The dependencies and assumptions of the interfaces holds only on the objects that act upon them.	The dependencies and assumptions of the all interfaces holds on all the components in the system.	They are integrated into the WWW.
Tightly coupled services	Tightly coupled services	Complete service that could also be made as a value added service

Communication paradigms

There are three major modes of communication in distributed systems:

- **Interprocess communication**:

  This is a low-level support for communication between processes in distributed systems, including message-passing primitives. They have direct access to the API offered by Internet protocols and support multicast communication.

- **Remote invocation:**

  It is used in a distributed system and it is the calling of a remote operation, procedure or method.

a) **Request-reply protocols:**

This is a message exchange pattern in which a requestor sends a request message to a replier system which receives and processes the request, ultimately returning a message in response. This is a simple, but powerful messaging pattern which allows two applications to have a two-way conversation with one another over a channel. This pattern is especially common in client–server architectures.

b) **Remote procedure calls:** Remote Procedure Call (**RPC**) is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details.

c) **Remote method invocation**: RMI (Remote Method Invocation) is a way that a programmer can write object-oriented programming in which objects on different computers can interact in a distributed network. This has the following two key features:

- ✓ Space uncoupling: Senders do not need to know who they are sending to
- ✓ Time uncoupling: Senders and receivers do not need to exist at the same time

➢ **Indirect communication**

Key techniques for indirect communication include:

a) **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. A group identifier uniquely identifies the group. The recipients join the group and receive the messages. Senders send messages to the group based on the group identifier and hence do not need to know the recipients of the message.

b) **Publish-subscribe systems: This** is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers. Instead, published messages are characterized into classes, without knowledge of what, subscribers there may be. Similarly, subscribers express interest in one or more classes, and only receive messages that are of interest, without knowledge of what, if any, publishers there are. They offer one-to-many style of communication.

c) **Message queues:** They 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. Queues therefore offer an indirection between the producer and consumer processes.

d) **Tuple spaces:** The 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. This style of programming is known as **generative communication**.

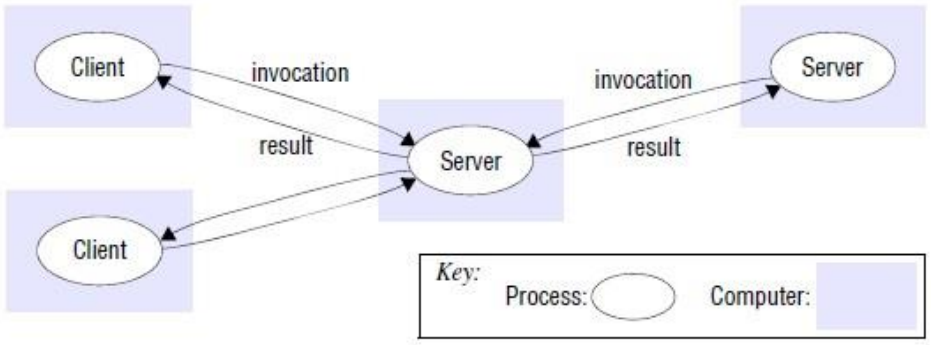
e) **Distributed shared memory:** In computer architecture, distributed shared memory (DSM) is a form of memory architecture where the (physically separate) memories can be addressed as one (logically shared) address space. Here, the term shared does not mean that there is a single centralized memory but shared essentially means that the address space is shared (same physical address on two processors refers to the same location in memory

Roles and responsibilities

There are two types of architectures based on roles and responsibilities:

1) **Client-server:**

- ✓ 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 primitives or using remote procedure calls (RPC) or remote method invocation (RMI).



for some
requested)
this new
ng in poor
ne server.

Figure 2.1: Client server style

2) Peer-peer

- ✓ All processes (objects) play similar role.
- ✓ Processes (objects) interact without particular distinction between clients and servers.
- ✓ The pattern of communication depends on the particular application.
- ✓ A large number of data objects are shared; any individual computer holds only a small part of the application database.

- ✓ Processing ts are distributed across mar
- ✓ This is the

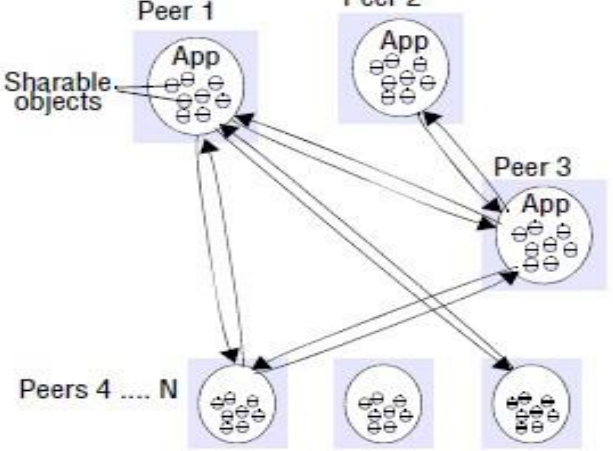


Figure 2.2: Peer-peer style

- ✓ Peer-to-Peer model distributes shared resources widely share computing and communication loads.
- ✓ Problems with peer-to-peer are high complexity due to cleverly place individual objects and there are large number of replicas.

Placement

Mapping of objects or services to the underlying physical distributed infrastructure is considered here. The following points must be taken care while placing the objects: reliability, communication pattern, load, QOS etc. The following are the placement strategies:

- ➢ **Mapping of services to multiple servers**
  - ✓ The servers may partition the set of objects on which the service is based and distribute those objects between themselves, or they may maintain replicated copies of them on several hosts.
  - ✓ 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.
- ➢ **Caching**
  - ✓ A cache is a local store of recently used data objects that is closer to one client or a particular set of clients than the objects themselves.
  - ✓ When a new object is received from a server it is added to the local cache store, replacing some existing objects if necessary.

- ✓ When an object is needed by a client process, the caching service first checks the cache and supplies the object from there if an up-to-date copy is available. If not, an up-to-date copy is fetched.
- ✓ Caches may be co-located with each client or they may be located in a proxy server that can be shared by several clients.

➢ **Mobile code:**

- ✓ Applets are a well-known and widely used example of mobile code.
- ✓ The user running a browser selects a link to an applet whose code is stored on a web server; the code is downloaded to the browser and runs.
- ✓ They provide interactive response.

➢ **Mobile agents:**

- ✓ A mobile agent is a running program (including both code and data) that travels from one computer to another in a network carrying out a task on someone‟s behalf, such as collecting information, and eventually returning with the results.
- ✓ A mobile agent is a complete program, code + data, that can work (relatively) independently.
- ✓ The mobile agent can invoke local resources/data.
- ✓ A mobile agent may make many invocations to local resources at each site it visit.
- ✓ Typical tasks of mobile agent includes: collect information , install/maintain software on computers, compare prices from various vendors bay visiting their sites etc.
- ✓ Mobile agents (like mobile code) are a potential security threat to the resources in computers that they visit.

2.2.2.2 Architectural patterns

Architectural patterns are reusable solution to a commonly occurring problem in software architecture within a given context. They are not complete solutions but rather offer partial insights. The following are some of the common architectural patterns:

➢ **Layering:**

- ✓ In a layered approach, a complex system is partitioned into a number of layers, with a given layer making use of the services offered by the layer below.

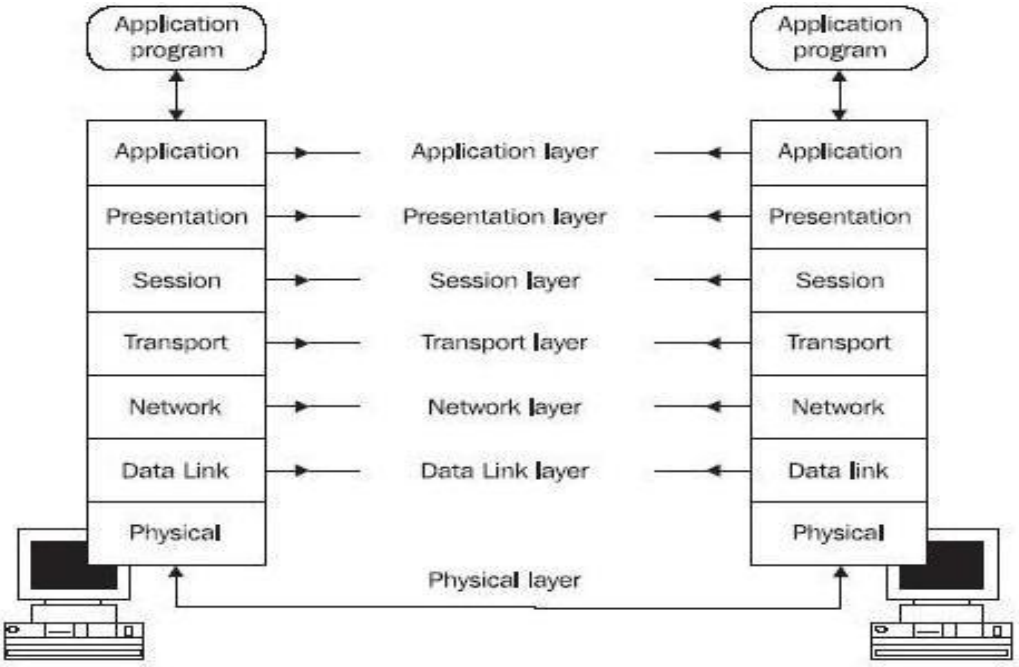


Figure 2.3: Layered Architecture

- ✓ A given layer is unaware of the implementation details, of other layers beneath them.
- ✓ In terms of distributed systems, this equates to a vertical organization of services into service layers.
- ✓ A distributed service can be provided by one or more server processes, interacting with each other and with client processes in order to maintain a consistent system-wide view of the service‟s resources.

➢ **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 two tiered architecture refers to client/server architectures in which the user interface (**presentation layer)** runs on the client and the database (**data layer**) is stored on the server. The actual application logic can run on either the client or the server.

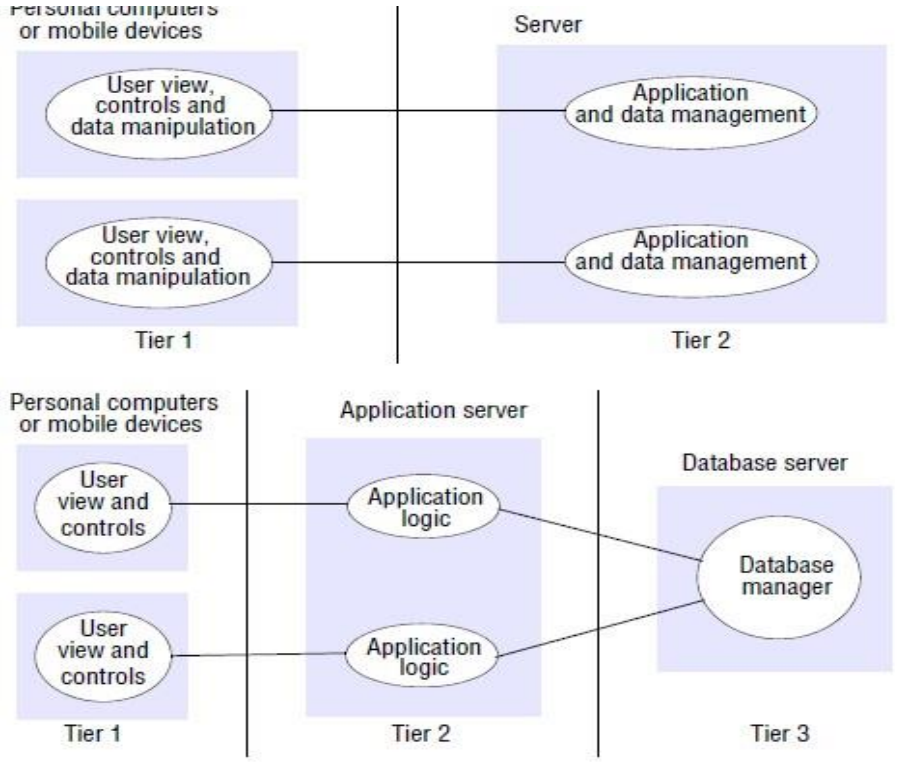


Fig 2.4: Two and three tiered architectures

- ✓ The three tiered architecture has:
  - **a) Presentation tier:** This is the topmost level of the application. is concerned with handling user interaction and updating the view of the application as presented to the user;
  - **b) Application tier (business logic, logic tier, or middle tier):** The logical tier is pulled out from the presentation tier and, as its own layer, it controls an application‟s functionality by performing detailed processing.
  - **c) Data tier:** The data tier includes the data persistence mechanisms (database servers, file shares, etc.) and the data access layer that encapsulates the persistence mechanisms and exposes the data.

➢ **Thin clients**

- ✓ In a thin-client model, all of the application processing and data management is carried out on the server.
- ✓ The client is simply responsible for running the responsible for running the presentation software.
- ✓ This is used when legacy systems are migrated to client server architectures.

- ✓ The legacy system acts as a server in its own right with a graphical interface implemented on a client.
- ✓ The major disadvantages of thin clients is that it places a heavy processing load on both the server and the network and the delay due to operating system latencies.
- ✓ The **virtual network computing**(VNC) has emerged to overcome the disadvantages og thin clients.
- ✓ VNC is a type of remote-control software that makes it possible to control another computer over a network connection.
- ✓ Keystrokes and mouseclicks are transmitted from one computer to another, allowing technical support staff to manage a desktop, server, or other networked device without being in the same physical location.
- ✓ Since all the application data and code is stored by a file server, the users may migrate from one network computer to another. So VNC is of big use in distributed systems.

➢ **Other patterns**

a) Proxy:

- ♦ This facilitates location transparency in remote procedure calls or remote method invocation.
- ♦ A proxy is created in the local address space to represent the remote object with same interface as the remote object.
- ♦ The programmer makes calls on this proxy object and he need not be aware of the distributed nature of the interaction.

b) Brokerage:

- ♦ It is used to bring interoperability in potentially complex distributed infrastructures.
- ♦ The service broker is meant to be a registry of services, and stores information about what services are available and who may use them.

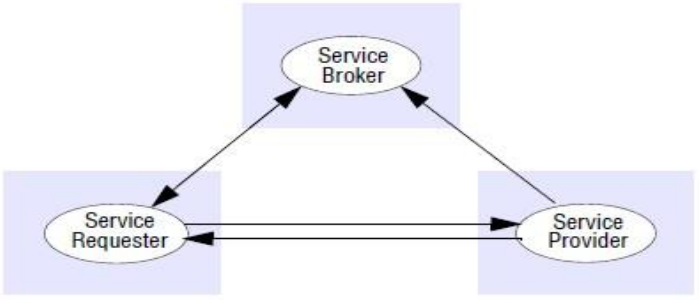


Fig 2.5: Web service with brokers

c) Reflection:

- The Reflection architectural pattern provides a mechanism for changing structure and behavior of software systems dynamically.
- It supports the modification of fundamental aspects, such as type structures and function call mechanisms.
- In this pattern, an application is split into two parts:

  1) **Meta Level:** This provides information about selected system properties and makes the software self-aware.
  2) **Base level:** This includes the application logic. Its implementation builds on the meta level. Changes to information kept in the meta level affect subsequent base-level behavior.

2 2.2.3 Associated middleware solutions

Middleware is a general term for software that serves to "glue together" separate, often complex and already existing, programs.

Middleware provides a higher-level programming abstraction for the development of distributed systems. Middleware makes it easier for software developers to perform communication and input/output, so they can focus on the specific purpose of their application. Middleware is the software that connects software components or enterprise applications and it lies between the operating system and the applications on each side of a distributed computer network. Middleware includes Web servers, application servers, content management systems, and similar tools that support application development and delivery.

Categories of middleware

The following are some of the categories of middleware:

- **Distributed Objects**

  The term distributed objects usually refers to software modules that are designed to work together, but reside either in multiple computers connected via a network or in different processes inside the same computer. One object sends a message to another object in a remote machine or process to perform some task. The results are sent back to the calling object.

- **Distributed Components**

  A component is a reusable program building block that can be combined with other components in the same or other computers in a distributed network to form an application. **Examples**: a single button in a graphical user interface, a small interest calculator, an interface to a database manager. Components can be

deployed on different servers in a network and communicate with each other for needed services. A component runs within a context called a container . **Examples:** pages on a Web site, Web browsers, and word processors.

- **Publish subscriber model**

  Publish–subscribe is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers. Instead, published messages are characterized into classes, without knowledge of what, if any, subscribers there may be. Similarly, subscribers express interest in one or more classes, and only receive messages that are of interest, without knowledge of what publishers are there.

- **Message Queues**

  Message queues provide an asynchronous communications protocol, meaning that the sender and receiver of the message do not need to interact with the message queue at the same time. Messages placed onto the queue are stored until the recipient retrieves them. Message queues have implicit or explicit limits on the size of data that may be transmitted in a single message and the number of messages that may remain outstanding on the queue.

- **Web services**

  A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

- **Peer to peer**

  Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or work load between peers. Peers are equally privileged, equipotent participants in the application. They are said to form a peer-to-peer network of nodes.

Advantages of middleware

- ✓ Real time information access among systems.
- ✓ Streamlines business processes and helps raise organizational efficiency.
- ✓ Maintains information integrity across multiple systems.
- ✓ It covers a wide range of software systems, including distributed Objects and components, message-oriented communication, and mobile application support.
- ✓ Middleware is anything that helps developers create networked applications.

Disadvantages of middleware

- ✓ Prohibitively high development costs.
- ✓ There are only few people with experience in the market place to develop and use a middleware.
- ✓ There are only few satisfying standards.
- ✓ The tools are not good enough.
- ✓ Too many platforms to be covered.
- ✓ Middleware often threatens the real-time performance of a system.
- ✓ Middleware products are not very mature.

2.2.3 Fundamental Models

Fundamental Models are concerned with a formal description of the properties that are common in all of the architectural models.

Models addressing time synchronization, message delays, failures, security issues are addressed as:

- ➢ **Interaction Model** – deals with performance and the difficulty of setting of time limits in a distributed system.
- ➢ **Failure Model** – specification of the faults that can be exhibited by processes
- ➢ **Secure Model** – discusses possible threats to processes and communication channels.

The purpose of the fundamental model is to:

- ♦ make explicit all the relevant assumptions about the systems we are modelling.
- ♦ make generalizations (general purpose algorithms) concerning what is possible or impossible with given assumptions.

2.2.3.1 Interaction model

- ✓ In this model the computations occurs within processes.
- ✓ The processes interact by passing messages to achieve communication (information flow) and coordination (synchronization and ordering of activities) between processes.

- ✓ The two significant factors affecting interacting processes in a distributed system are:

  1) Communication performance

  2) Global notion of time.

Communication Performance

- The communication channel in can be implemented in a variety of ways in distributed systems: Streams or through simple message passing over a network.
- The performance characteristics of a network are:

  a) **Latency:** A delay between the start of a message‟s transmission from one process to the beginning of reception by another.

  b) **Bandwidth:** The total amount of information that can be transmitted over in a given time. The communication channels using the same network, have to share the available bandwidth.

  c) Jitter: The variation in the time taken to deliver a series of messages. It is very relevant to multimedia data.

Global Notion of time

- Each computer in a distributed system has its own internal clock, which can be used by local processes to obtain the value of the current time.
- Any two processes running on different computers can associate timestamp with their events.
- However, even if two processes read their clocks at the same time, their local clocks may supply different time because the computer clock drifts from perfect time and their drift rates differ from one another.
- Even if the clocks on all the computers in a distributed system are set to the same time initially, their clocks would eventually vary quite significantly unless corrections are applied.
- There are several techniques to correct time on computer clocks.
- One technique is to use radio receivers to get readings from GPS (Global Positioning System) with accuracy about 1 microsecond**.**

Variants of Interaction model

- ✓ In a DS it is hard to set time limits on the time taken for process execution, message delivery or clock drift. There are two models based on time stamp:

- **Synchronous DS**
  - This is practically hard to achieve in real life.
  - In synchronous DS the time taken to execute a 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 known bound.
- **Asynchronous DS**
  - There are no bounds on: process execution speeds, message transmission delays and clock drift rates.

Event Modeling

- ✓ Event ordering is of major concern in DS.
- ✓ The concept of one event happening before another in a distributed system is examined, and is shown to define a partial ordering of the events.
- ✓ The execution of a system can be described in terms of events and their ordering despite the lack of accurate clocks.
- ✓ Consider a mailing list with users X, Y, Z, and A.
- ✓ Due to independent delivery in message delivery, message may be delivered in different order.
- ✓ If messages m1, m2, m3 carry their time t1, t2, t3, then they can be displayed to users accordingly to their time ordering.
- ✓ The mail box is:

Item	**From**	**Subject**
23	Z	Re: Meeting
24	X	Meeting
26	Y	Re: Meeting

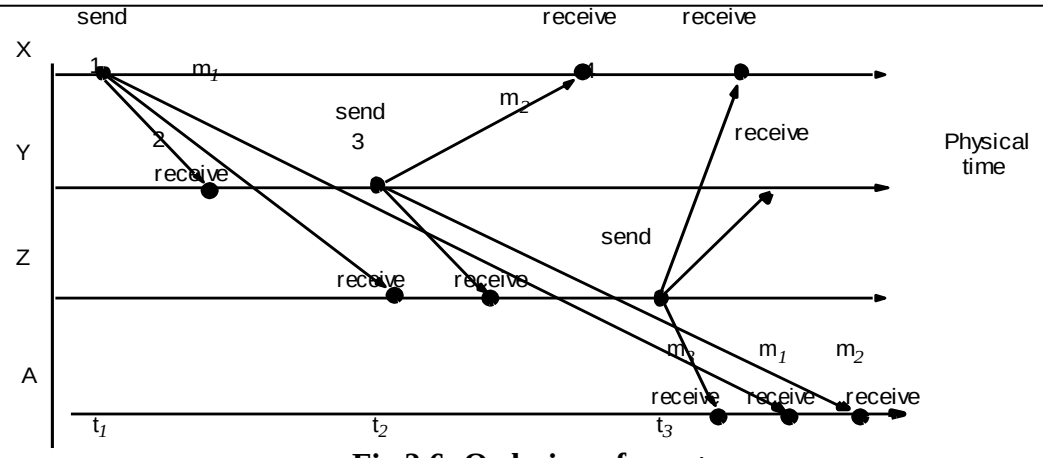


Fig 2.6: Ordering of events

2.2.3.2 Failure Model

In a DS, both processes and communication channels may fail – i.e., they may depart from what is considered to be correct or desirable behavior. The following are the common types of failures:

- Omission Failure
- Arbitrary Failure
- Timing Failure

a) Omission failure

Omission failures occur due to communication link failures. They are detected through timeouts. They are classified as:

- **Process omission failures**
  - ✓ Omission failures that occur due to process crash (i.e.) the execution of the process could not continue.
  - ✓ This is detected using timeouts.
  - ✓ A fixed period of time is fixed for all the methods to complete its execution. If the method takes time longer than the allowed time, a time out has occurred.
  - ✓ In an asynchronous system a timeout can indicate only that a process is not responding.
  - ✓ A process crash is called **fail-stop**if other processes can detect certainly that the process has crashed.

- ✓ Fail-stop behavior can be produced in a synchronous system if the processes use timeouts to detect when other processes fail to respond and messages are guaranteed to be delivered.

➢ **Communication omission failures**

- ✓ This occurs when the messages are dropped between sender and the receiver.
- ✓ The message can be lost in sender buffer, receiver buffer or even in the communication channel.
- ✓ The loss of messages between the sending process and the outgoing message buffer is known as **send omission failures.**
- ✓ The loss of messages between the incoming message buffer and the receiving process as **receive-omission failures.**
- ✓ The loss of messages in a channel is **channel omission failure.**

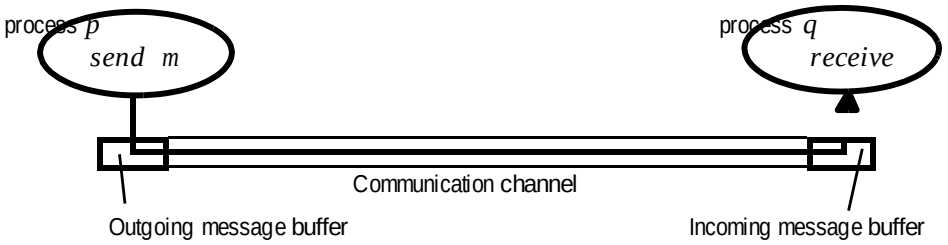


Fig 2.7: Communication between processes in a channel

b) **Arbitrary failures (Byzantine failure):**

- ✓ This includes all possible errors that could cause failure.
- ✓ Communication channels often suffer from arbitrary failures
- ✓ The following table gives an overview of the different failures and its categories:

Class	**Affects**	**Comments**
Fail stop	Process	Process halts and remains halted. Other processes may detect that this process has halted.
Crash	Process	Process halts and remains halted. Other processes may not detect that this process has halted.

Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end‟s incoming messagebuffer.
Send-omission	Process	A process completes a send operation but the message is not put in its outgoing message buffer.
Receiveomission	Process	A message is put in a process‟s incoming messagebuffer, but that process does not receive it.
Arbitrary	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times or commit omissions; a process may stop or take an incorrect step.

c) **Timing failures**

- ✓ Timing failures refer to a situation where the environment in which a system operates does not behave as expected regarding the timing assumptions, that is, the timing constraints are not met.
- ✓ Timing failures are applicable in synchronous distributed system where time limits are set on process execution time, message delivery time and clock drift rate.
- ✓ The following are the common failures withnrespect to timings:

Class	**Affects**	**Comments**
Clock	Process	Process‟s local clock exceeds the bounds on its rate of drift from real time
Performance	Process	Process exceeds the bounds on the interval between two steps
Performance	Channel	A message‟s transmission takes longer than the stated bound.

Masking failures

- ✓ Each component in a distributed system is generally constructed from a collection of other components. It is always possible to construct reliable services from components that exhibit failures.
- ✓ A knowledge of failure characteristics of a component can enable a new service to be designed to mask the failure of the components on which it depends.

Reliability of one-to-one communication

The term **reliable communication** is defined in terms of validity and integrity.

- **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.

The threats to integrity come from two independent sources:

- Any protocol that retransmits messages but does not reject a message that arrives twice.
- Malicious users that may inject spurious messages, replay old messages or tamper with messages.

2.2.3.2 Security model

The security of a DS can be achieved by securing the processes and the channels used in their interactions and by protecting the objects that they encapsulate against unauthorized access.

Protection is described in terms of objects, although the concepts apply equally well to resources of all types.

Protecting objects

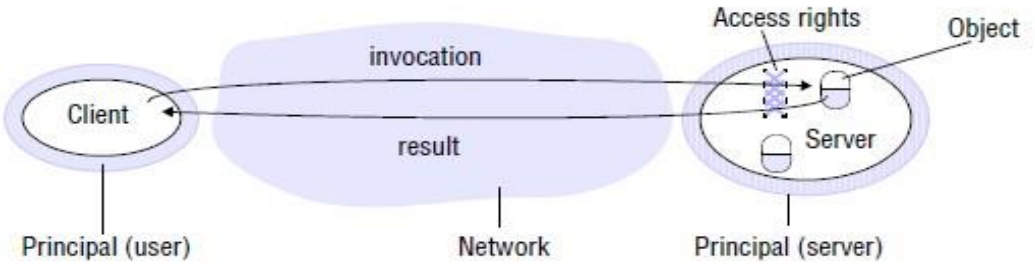


Fig 2.8: Objects and Principals

- ✓ The server manages a collection of objects on behalf of some users.
- ✓ The users can run client programs that send invocations to the server to perform operations on the objects.
- ✓ The server carries out the operation specified in each invocation and sends the result to the client.

- ✓ This uses use "access rights" that define who is allowed to perform operation on a object.
- ✓ The server should verify the identity of the principal (user) behind each operation and checking that they have sufficient access rights to perform the requested operation on the particular object, rejecting those who do not.
- ✓ A principal is an authority that manages the access rights. The principal may be a user or a process.

Securing processes and their interactions

- ✓ The messages send by the processes are exposed to attack because the network and the co
- ✓ To m nding any proce
- ✓ Threa hreats to comm

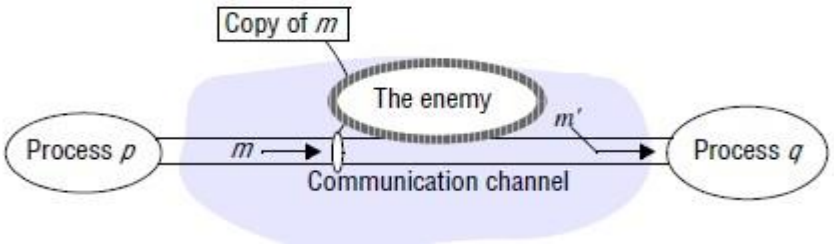


Fig: 2.9: Enemy

- ➢ **Threats to processes:**
  - ♦ A process that is designed to handle incoming requests may receive a message from any other process in the distributed system
  - ♦ It cannot necessarily determine the identity of the sender.
  - ♦ This lack of reliable knowledge of the source of a message is a threat to the correct functioning of both servers and clients.
- ➢ **Servers**:
  - ♦ Servers cannot necessarily determine the identity of the principal behind any particular invocation.
  - ♦ Without reliable knowledge of the sender"s identity, a server cannot tell whether to perform the operation or to reject it.

- **Clients:**
  - When a client receives the result of an invocation from a server, it cannot necessarily tell whether the source of the result message is from the intended server or from an enemy, perhaps „spoofing" the mail server.
  - Thus the client could receive a result that was unrelated to the original invocation, such as a false mail item (one that is not in the user"s mailbox).
- **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.
- **Defeating security threats**
  - Encryption and authentication are use to build secure channels.
  - Each of the processes knows the identity of the principal on whose behalf the other process is executing and can check their access rights before performing an operation.

Other possible threats from an enemy

- **Denial of service:**

  Denial of service (DoS) attack is an incident in which a user or organization is deprived of the services of a resource they would normally expect to have. In a distributed denial-of-service, large numbers of compromised systems (sometimes called a botnet) attack a single target.

- **Mobile code:**

  Mobile code is software transferred between systems, e.g. transferred across a network, and executed on a local system without explicit installation by the recipient. Mobile code raises new and interesting security problems for anyprocess that receives and executes program code from elsewhere.

The uses of security models

The use of security techniques incurs processing and management costs. The distributed systems faces threats from various points. The threat analysis demands the construction of security models.

2.3 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.

- IPC refers to exchange of data between two or more separate, independent processes/threads.
- The operating systems provide facilities/resources for inter-process communications (IPC), such as message queues, semaphores, and shared memory.
- Distributed computing systems make use of these facilities/resources to provide application programming interface (API) which allows IPC to be programmed at a higher level of abstraction. (e.g., send and receive)
- Distributed computing requires information to be exchanged among independent processes.

2.3.1 The API's for the Internet Protocols

Characteristics of IPC:

- **Synchronous and asynchronous communication**
  - ✓ In the synchronous form of communication, the sending and receiving processes synchronize at every message.
  - ✓ In this case, both send and receive are blocking operations.
  - ✓ In the asynchronous form of communication, the use of the send operation is non blocking in that 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.
  - ✓ The receive operation can have blocking and non-blocking variants.
- **Message destinations**
  - ♦ The messages in the Internet protocols, messages are sent to (Internet address, local port) pairs.
  - ♦ A local port is a message destination with in a computer, specified as an integer.
  - ♦ A port has exactly one receiver but can have many senders.
  - ♦ Processes may use multiple ports to receive messages.
  - ♦ Any process that knows the number of a port can send a message to it.
  - ♦ Client programs refer to services by name and use a name server or binder to translate their names into server locations at runtime.
  - ♦ This allows services to be relocated but not to migrate – that is, to be moved while the system is running.

- **Reliability**
  - A point-to-point message service can be described as reliable if messages are guaranteed to be delivered.
  - A point-to-point message service can be described as unreliable if messages are not guaranteed to be delivered in the face of even a single packet dropped or lost.
- **Ordering**
  - Some applications require that messages be delivered in the order in which they were transmitted by the sender.
  - The delivery of messages outof sender order is regarded as a failure by such applications.

Sockets

A socket is one endpoint of a two-way communication link between two programs running on the network.

Interprocess communication consists of transmitting a message between a socket in one process and a socket in another process. The java API for interprocess communication in the internet provides both datagram and stream communication. Both forms of communication, UDP and TCP, use the socket abstraction, which provides and endpoint for communication between processes. Interprocess communication consists of transmitting a message between a socket in one process and a socket in another process. The two communication patterns that are most commonly used in distributed programs:

- Client-Server communication
  - The request and reply messages provide the basis for remote method invocation (RMI) or remote procedure call (RPC).
- Group communication
  - The same message is sent to several processes.

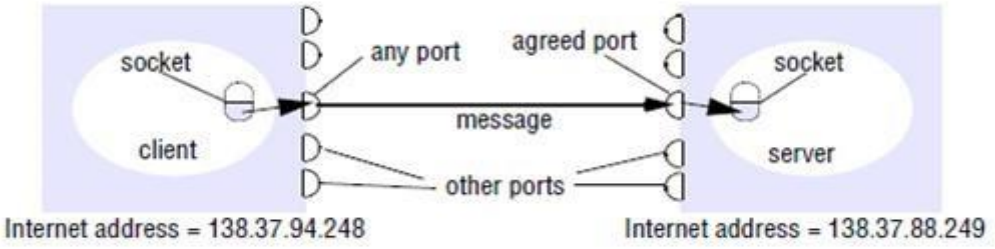


Fig 2.6: Sockets and ports

For a process to receive messages, its socket must be bound to a local port and one of the Internet addresses of the computer on which it runs. Messages sent to a particular Internet address and port number can be received only by a process whose socket is associated with that Internet address and port number.

Java API for Internet addresses

- ✓ Java provides a class, InetAddress, that represents Internetaddresses.
- ✓ Users of this class refer to computers by Domain Name System (DNS) hostnames. InetAddressaComputer = InetAddress.getByName("abc.ac.in");

UDP Datagram Communication

- The application program interface to UDP provides a message passing abstraction.
- Message passing is the simplest form of interprocess communication.
- API enables a sending process to transmit a single message to a receiving process.
- The independent packets containing theses messages are called **datagrams.**
- In the Java and UNIX APIs, the sender specifies the destination using a socket.
- A datagram is transmitted between processes when one process sends it and another receives it.
- The following are the steps in socket communication:
  - Creating a socket
  - Binding a socket to a port and local Internet address
    - A client binds to any free local port
    - A server binds to a server port
- Receive() returns Internet address and port of sender, along with the message.

Issues in datagram communication:

- **Message size:**
  - ✓ The receiving process needs to specify an fixed size array to receive a message.
  - ✓ If the message is too big for the array, it is truncated.
  - ✓ Any application requiring messages larger than the maximum must fragment them.

- **Blocking:**
  - ✓ Sockets normally provide non-blocking sends and blocking receives for datagram communication.
  - ✓ The send() returns after delivering the message to theUDP and IP protocols. These protocols are now responsible for transmitting the message to its destination
  - ✓ The message is placed in a queue for the socket that is bound to the destination port.
  - ✓ Messages are discarded atthe destination if no process has a socket bound to the destination port.
  - ✓ The receive()blocks until a datagram is received, unless a timeout has been set on the socket.
- **Timeouts:**
  - ✓ The receive() cannot wait indefinitely. This situation occurs when the sending process may have crashed or the expected message may have been lost.
  - ✓ To avoid this timeouts can be set on sockets.
  - ✓ The timeouts must be much larger than the time required to transmit a message.
- **Receive from any:**
  - ✓ The receive() does not specify an origin for messages. This can get datagrams addressed to its socket from any origin.
  - ✓ The receive() returns the Internet address and local port of the sender, allowing the recipient to check where the message came from.
  - ✓ It is possible to connect a datagram socket to a particular remote port and Internet address, in which case the socket is only able to send messages to and receive messages from that address.

Failure model for UDP datagrams

UDP datagrams suffer from following failures:

- Omission failure: Messages may be dropped occasionally
- Ordering: Messages can be delivered out of order.

Java API for UDP datagrams

The Java API provides datagram communication by two classes:

1. DatagramPacket
   - It provides a constructor to make an array of bytes comprising:
     - Message content
     - Length of message
     - Internet address
     - Local port number
   - It provides another similar constructor for receiving a message.
2. DatagramSocket
   - This class supports sockets for sending and receiving UDP datagram.
   - It provides a constructor with port number as argument.
   - No-argument constructor is used to choose a free local port.
   - DatagramSocket methods are:
     1. **send and receive:**These methods are used for transmitting datagrams between a pair of sockets.

        Syntax:

        send(packet);

        receive();

     2. **setSoTimeout:** This method allows a timeout to be set. With a timeout set, the receive method will block for the time specified and then throw an InterruptedIOException.

        Syntax:

        setSotimeout(time in seconds);

     3. **connect:** This method is used for connecting to a particular remote port and Internet address, in which case the socket is only able to send messages to and receive messages from that address.

        Syntax:

        int connect(in sockfd, const structsockaddr *addr, socklen_taddrlen);

Arrays of bytes containing message	Message length	Internet address	Port number

Fig 2.7: Datagram Packet

UDP Client server Communication

Client Program

```
import java.net.*;
import java.io.*;
public class UDPClient
{
public static void main(String args[])
{
DatagramSocketaSocket = null;
        try {
aSocket = new DatagramSocket(); // Create an object for the datagram class
                byte [] m = args[0].getBytes(); //Buffer to get data
InetAddressaHost = InetAddress.getByName(args[1]);  // Get Inetaddress
intserverPort = 6789;
DatagramPacket request =
                    new DatagramPacket(m, m.length(), aHost, serverPort);
aSocket.send(request); //Sent the packet
   byte[] buffer = new byte[1000];
DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
aSocket.receive(reply);
System.out.println("Reply: " + new String(reply.getData()));
              }
       catch (SocketException e){System.out.println("Socket: " + e.getMessage());
    }
 catch (IOException e){System.out.println("IO: " + e.getMessage());
}
```

```
 finally
  {
       if(aSocket != null) aSocket.close();}
   }
}
```

Server program

```
import java.net.*;
import java.io.*;
public class UDPServer
{
public static void main(String args[])
{
DatagramSocketaSocket = null;
    try
   {
aSocket = new DatagramSocket(6789);
          byte[] buffer = new byte[1000];
          while(true)
           {
DatagramPacket request = new DatagramPacket(buffer, buffer.length);
aSocket.receive(request);
DatagramPacket reply = new DatagramPacket(request.getData(),
request.getLength(), request.getAddress(), request.getPort());
aSocket.send(reply);
           }
    }
catch (SocketException e)
{
System.out.println("Socket: " + e.getMessage());
```

```
 }
 catch (IOException e)
{
System.out.println("IO: " + e.getMessage());
}
finally
{
if (aSocket != null) aSocket.close();}
}
}
```

TCP stream Communication

The API to the TCP protocol provides the abstraction of a stream of bytes to be written to or read from. The following features are hidden stream abstraction:

- Message sizes
- Lost messages
- Flow control
- Message destinations

Working of stream communication

- ✓ 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.
- ✓ One of the pair of processes can send information to the other by writing to its output stream, and the other process obtains the information by reading from its input 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 flushed to the other end of the stream and put in the queue at the destination socket, with an indication that the socket is closed.
- ✓ The process at the destination can read the data in the queue, but any further reads after the queue is empty will result in an indication of end of stream.
- ✓ 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.

This form of stream communication has the following issues:

- **Matching of data items:** Two communicating processes need to agree as to the contents of the data transmitted over a stream.
- **Blocking:** The process that writes data to a stream may be blocked by the TCP flow-control mechanism if the socket at the other end is queuing as much data as the protocol allows.
- **Threads:** When a server accepts a connection, it generally creates a new thread in which to communicate with the new client. In an environment in which threads are not provided, an alternative is to test whether input is available from a stream before attempting to read it;

Java API for TCP streams

The Java interface to TCP streams is provided in the classes:

- **ServerSocket:** It is used by a server to create a socket at server port to listen for connect requests from clients.

  Syntax:

  publicServerSocket(int port, InetAddress address) ;

- **Socket:** It is used by a pair of processes with a connection. The client uses a constructor to create a socket and connect it to the remote host and port of a server.It provides methods for accessing input and output streams associated with a socket.

  Syntax:

  public Socket(InetAddress host, int port, InetAddresslocalAddress, intlocalPort);

TCP Client Server Communication

TCP Client

```
import java.net.*;
import java.io.*;
public class TCPClient
{
public static void main (String args[])
{
    Socket s = null;
```

```
    try
    {
intserverPort = 7896;
          s = new Socket(args[1], serverPort);
DataInputStream in = new DataInputStream( s.getInputStream());
DataOutputStream out = new DataOutputStream( s.getOutputStream());
out.writeUTF(args[0]); // UTF is a string encoding;
          String data = in.readUTF();
System.out.println("Received: "+ data) ;
      }
     catch (UnknownHostException e)
  {
System.out.println("Sock:"+e.getMessage());
  }
catch (EOFException e)
{       System.out.println("EOF:"+e.getMessage());
}
catch (IOException e)
{        System.out.println("IO:"+e.getMessage());
}
finally
{       if(s!=null)
try
{
s.close();
}
catch (IOException e)
{
```

```
/*close failed*/
}
}
}
}
```

TCP server

```
import java.net.*;
import java.io.*;
public class TCPServer
{
public static void main (String args[])
{
try{
intserverPort = 7896;
ServerSocketlistenSocket = new ServerSocket(serverPort);
while(true) {
Socket clientSocket = listenSocket.accept();
Connection c = new Connection(clientSocket);
}
} catch(IOException e) {System.out.println("Listen :"+e.getMessage());}
}
}
class Connection extends Thread
{
DataInputStream in;
DataOutputStream out;
Socket clientSocket;
```

```
public Connection (Socket aClientSocket)
{
try {
clientSocket = aClientSocket;
in = new DataInputStream( clientSocket.getInputStream());
out =new DataOutputStream( clientSocket.getOutputStream());
this.start();
} catch(IOException e) {System.out.println("Connection:"+e.getMessage());}
}
public void run(){
try { // an echo server
String data = in.readUTF();
out.writeUTF(data);
} catch(EOFException e) {System.out.println("EOF:"+e.getMessage());
} catch(IOException e) {System.out.println("IO:"+e.getMessage());
} finally { try {clientSocket.close();}catch (IOException e){/*close failed*/}}
}
}
```

2.4 EXTERNAL DATA REPRESENTATION AND MARCHALLING

The information stored in running programs is represented as data structures, whereas the information in messages is in the form of sequences of bytes. Irrespective of the form of communication used, the data structure must be converted to a sequence of bytes before transmission and must be rebuilt on arrival.

External Data Representation is an agreed standard for the representation of data structures and primitive values.

But there are some problems in data representation. They are:

- Two conversions are necessary (i.e.) conversion of bytes to data structures and vice versa in case of using an agreed format at both ends.
- Using sender‟s or receiver‟s format and convert at the other end.

Marshalling is the process of taking a collection of data items and assembling them into a form suitable for transmission in a message. Unmarshalling is the process of disassembling a collection of data on arrival to produce an equivalent collection of data items at the destination.

The following are the three common approaches to external data representation and marshalling:

- **CORBA:** The data representation is concerned with an external representation for the structured and primitive types that can be passed as the arguments and results of remote method invocations in CORBA. It can be used by a variety of programming languages. Marshalling and unmarshalling is done by the middleware in binary form.
- **Java's object serialization:** This is concerned with the flattening and external data representation of any single object or tree of objects that may need to be transmitted in a message or stored on a disk. It is for use only by Java. Marshalling and unmarshalling is done by the middleware in binary form.
- **XML**: This defines a textual fomat for representing structured data. It was originally intended for documents containing textual self-describing structured data. Marshalling and unmarshalling is in texual form.

2.4.1 CORBA'a Common Data Representation (CDR)

CORBA CDR is the external data representation defined with CORBA 2.0.It consists 15 primitive types:

- ✓ Short (16 bit)
- ✓ Long (32 bit)
- ✓ Unsigned short
- ✓ Unsigned long
- ✓ Float(32 bit)
- ✓ Double(64 bit)
- ✓ Char
- ✓ Boolean(TRUE,FALSE)
- ✓ Octet(8 bit)
- ✓ Any(can represent any basic or constructed type)

There are two types of CDR in CORBA:

- **Primitive types:** CDR defines a representation for both big-endian and little-endian orderings. Characters are represented by a code set agreed between client and server.
- **Constructed types**: The primitive values that comprise each constructed type are added to a sequence of bytes in a particular order. The following are the constructed types:
  - ✓ sequence length (unsigned long): followed by elements in order
  - ✓ string length (unsigned long): followed by characters in order (can also have wide characters)
  - ✓ array: array elements in order (no length specified because it is fixed)
  - ✓ struct: in the order of declaration of the components
  - ✓ enumerated unsigned long: the values are specified by the order declared
  - ✓ union: type tag followed by the selected member

Example:

struct with value {„Sona", „Chennai", 1986}

0-3	4	Length of the string
4-7	"Sona"	String
8-11	"@_____"	
12-15	7	Length of the string
16-19	"Chen"	Chennai
20-23	"on____"	
24-27	1986	Unsigned long

Marshalling in CORBA

The type of a data item not given in case of CORBA. CORBA assumes that both the sender and recipient have common knowledge of the order and types of data items. The types of data structures and types of basic data items are described in CORBA IDL. This provides a notation for describing the types of arguments and results of RMI methods

Example:

```
Struct student
 {
string name;
stringrollnumber;
 }
```

2.4.2 Java Object Serilaization

In Java RMI, both object and primitive data values may be passed as arguments and results of method invocation. An object is an instance of a Java class. The Java equivalent for the above mentioned CORBA construct is :

```
Public class student implements Serializable
{          Private String name;
            Private String rollnumber;
            Public student(String aName ,String  arollnumber)
           {
          name = aName;
          rollnumber= arollnumber;
          }
}
```

Student	8 bit version number	h0	Class name, version
2	Java.lang.string.name	Java.lang.string.rollnumber	Number, type and name of instance variables
5Sona	8CS50	h1	Values of variables

To serialize an object: ‹

(1) its class info is written out: name, version number ‹

(2) types and names of instance variables • If an instance variable belong to a new class, then new class info must be written out, recursively. Each class is given a handle ‹

(3) values of instance variables ‹

Example: student s = new student(“5Sona”, “8CS50”);

‹ Thefollowinga re the steps to serialize objects in Java:

- ✓ create an instance of ObjectOutputStream ‹
- ✓ Invoke writeObject method passing Person object as argument ‹

To deserialize

- ✓ create an instance of ObjectInputStream ‹
- ✓ Invoke readObject method to reconstruct the original object

```
ObjectOutputStream out = new ObjectOutputStream(… );
out.writeObject(originalStudent);
ObjectInputStream in = new ObjectInputStream(…);
Student theStudent = in.readObject();
```

Reflection:

Reflection is inquiring about class properties, e.g., names, types of methods and variables, of objects. This allows to perform serialization and deserialization in a generic manner, unlike in CORBA, which needs IDL specifications. For serialization, use reflection to find out

(1) class name of the object to be serialized

(2) the names, types

(3) values of its instance variables „

For deserialization,

(1) class name in the serialized form is used to create a class

(2) it is then used to create a constructor with arguments types corresponding to those specified in the serialized form.

(3) the new constructor is used to create a new object with instance variables whose values are read from the serialized form

2.4.3 XML (Extensible Markup Language)

- ✓ XML is a markup language that was defined by the World Wide Web Consortium (W3C) for general use on the Web.
- ✓ Markup language refers to a textual encoding that represents both a text and details as to its structure or its appearance.

- ✓ In XML, the data items are tagged with „markup" strings.
- ✓ The tags are used to describe the logical structure of the data and to associate attribute-value pairs with logical structures.
- ✓ XML is used to enable clients to communicate with web services and for defining the interfaces and other properties of web services.
- ✓ XML is extensible in the sense that users can define their own tags, in contrast to HTML, which uses a fixed set of tags.
- ✓ XML could be used by multiple applications for different purposes.

Example:

```
<student id="123456789">
<name>Sona</name>
<rollnumber>CS50</rollnumber>
<year>1984</year>
```

XML elements and attributes

- **Elements:** An element in XML consists of a portion of character data surrounded by matching start and end tags.
- **Attributes:** A start tag may optionally include pairs of associated attribute names and values such as id="123456789".
- **Names:** The names of tags and attributes in XML generally start with a letter, but can also start with an underline or a colon. The names continue with letters, digits, hyphens, underscores, colons or full stops. Letters are case-sensitive. Names that start with xml are reserved.
- **Binary data:** All of the information in XML elements must be expressed as character data. The encrypted elements or secure hashes are represented as base64 notation.
- **CDATA:** XML parsers normally parse the contents of elements because they may contain further nested structures. If text needs to contain an angle bracket or a quote, it may be represented in a special way.
- **Namespace:** A namespace applies within the context of the enclosing pair of start and end tags unless overridden by an enclosed namespace declaration.

2.4.3 Remote Object Reference

- ✓ Remote object references are used when a client invokes an object that is located on a remote server.

- ✓ A remote object reference is passed in the invocation message to specify which object is to be invoked.
- ✓ Remote object references must be unique over space and time.

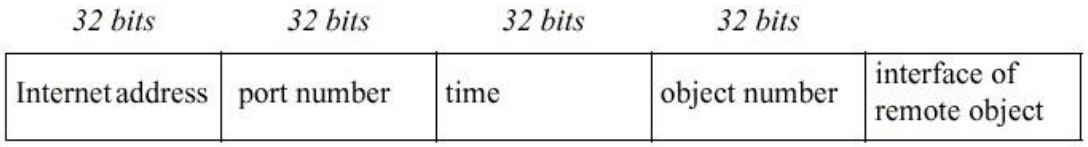


Fig 2.8: Representation of remote object references

The following are mandatory for remote object reference

- ➢ internet address/port number: process which created object
- ➢ time: creation time
- ➢ object number: local counter, incremented each time an object is created in the creating process
- ➢ interface: how to access the remote object (if object reference is passed from one client to another)

2.5 MULTICAST COMMUNICATION

Multicast (one-to-many or many-to-many distribution) is group communication where information is addressed to a group of destination computers simultaneously.

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.

The following are the characteristics of multicasting:

- ➢ Fault tolerance based on replicated services:
  - ✓ A replicated service consists of a group of servers.
  - ✓ Client requests are multicast to all the members of the group, each of which performs an identical operation.
- ➢ Finding the discovery servers in spontaneous networking
  - ✓ Multicast messages can be used by servers and clients to locate available discovery services in order to register their interfaces or to look up the interfaces of other services in the distributed system.

- Better performance through replicated data
  - ✓ Data are replicated to increase the performance of a service.
- Propagation of event notifications
  - ✓ Multicast to a group may be used to notify processes when something happens.

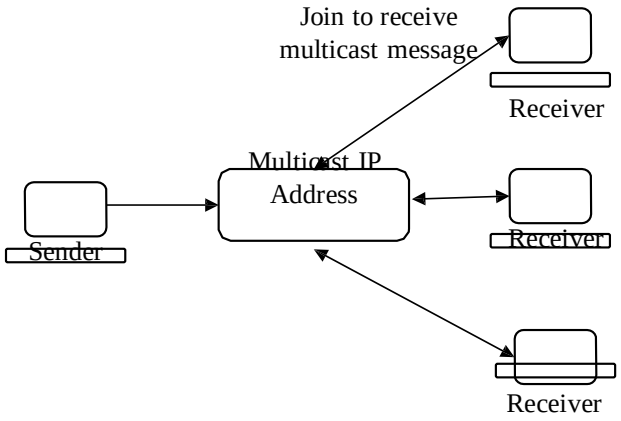


Fig 2.9: Multicasting

2.5.1 IP Multicast

- IP multicast is built on top of the Internet protocol.
- 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 with1110 as it starting byte.
- The membership of a multicast group is dynamic.
- A computer is said to be in a multicast group if one or more processes have sockets that belong to the multicast group.

The following details are important when using IPV4 IP multicasting:

- **Multicast IP routers**
  - IP packets can be multicast both on local network and on the Internet.
  - Local multicast uses local network such as Ethernet.
  - 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 (TTL).

- **Multicast address allocation**
  - Multicast addressing may be permanent or temporary.
  - Permanent groups exist even when there are no members.
  - Multicast addressing by temporary groups must be created before use and will stop to exit when all members have left the particular multicast group.
  - To start or to join a particular multicast group, a **session directory (sd)** program is used.
  - This is a tool with an interactive interface that allows users to browse advertised multicast sessions and to advertise their own session, specifying the time and duration that it wishes to be in the group.

Failure model for IP multicast datagram

They suffer from omission failures. The messages are not guaranteed to be delivered to the group member in case of omission failures.

2.5.2 Java API to IP multicast

- ✓ The Java API provides a datagram interface to IP multicast through the class MulticastSocket.
- ✓ MulticastSocket is a subset of DatagramSocket.
- ✓ It has the capability of being able to join multicast groups.
- ✓ The multicast datagram socket class is useful for sending and receiving IP multicast packets.
- ✓ A MulticastSocket is a (UDP) DatagramSocket, with additional capabilities for joining "groups" of other multicast hosts on the internet.
- ✓ The class MulticastSocket provides two constructors:
  - MulticastSocket(): Create a multicast socket.
  - MulticastSocket(int): Create a multicast socket and bind it to a specific port.

MulticastSender.java

```
import java.io.*;
import java.net.*;
public class MulticastSender {
```

```
public static void main(String[] args) {
  DatagramSocket socket = null;
  DatagramPacketoutPacket = null;
  byte[] outBuf;
  final int PORT = 8888;
   try {
    socket = new DatagramSocket();
    long counter = 0;
    String msg;
     while (true) {
      msg = " multicast! " + counter;
      counter++;
      outBuf = msg.getBytes();
       //Send to multicast IP address and port
      InetAddress address = InetAddress.getByName("224.2.2.3");
      outPacket = new DatagramPacket(outBuf, outBuf.length, address, PORT);
       socket.send(outPacket);
      System.out.println("Server sends : " + msg);
      try {
        Thread.sleep(500);
      } catch (InterruptedExceptionie) {
      }
    }
  } catch (IOExceptionioe) {
    System.out.println(ioe);
  }
 }
}
```

Multicastreceiver.java

```
import java.io.*;
import java.net.*;
public class MulticastReceiver {
  public static void main(String[] args) {
    MulticastSocket socket = null;
    DatagramPacketinPacket = null;
    byte[] inBuf = new byte[256];
    try {
      //Prepare to join multicast group
      socket = new MulticastSocket(8888);
      InetAddress address = InetAddress.getByName("224.2.2.3");
      socket.joinGroup(address);
       while (true) {
        inPacket = new DatagramPacket(inBuf, inBuf.length);
        socket.receive(inPacket);
        String msg = new String(inBuf, 0, inPacket.getLength());
        System.out.println("From " + inPacket.getAddress() + " Msg : " + msg);
      }
    } catch (IOExceptionioe) {
      System.out.println(ioe);
    }
  }
}
```

The following methods are available in MulticastSocket class:

- **getInterface**():Retrieve the address of the network interface used for multicast packets.
- **getTTL**():Get the default time-to-live for multicast packets sent out on the socket.

- **joinGroup**(InetAddress): Joins a multicast group.
- **leaveGroup**(InetAddress):Leave a multicast group.
- **send**(DatagramPacket, byte):Sends a datagram packet to the destination, with a TTL (time- to-live) other than the default for the socket.
- **setInterface**(InetAddress): Set the outgoing network interface for multicast packets on this socket, to other than the system default.
- **setTTL**(byte): Set the default time-to-live for multicast packets sent out on this socket.

2.5.2 Reliability and ordering in multicasting

There are many factors that puts the reliability of multicasting under question:

- ✓ A datagram sent from one multicast router to another may be lost.
- ✓ There are chances for a single datagram to arrive at multiple recipients, any one of those recipients may drop the message because its buffer is full.
- ✓ If a multicast router fails, the group members beyond that router will not receive the multicast message.

Ordering issue:

IP packets do not necessarily arrive in the order in which they were sent. Messages sent by two different processes will not necessarily arrive in the same order at all the members of the group. The problem of reliability and ordering is more common in:

- Fault tolerance based on replicated services
- Discovering services in spontaneous networking
- Better performance through replicated data
- Propagation of event notifications

2.6 NETWORK VIRTULIZATION: OVERLAY NETWORKS

Network virtualization is the construction of many different virtual networks over an existing network such as the Internet.

Network virtualization refers to the management and monitoring of an entire computer network as a single administrative entity from a single software-based administrator's console.

NV is implemented by installing software and services to manage the sharing of storage, computing cycles and applications. All servers and services in the network are treated as a single pool of resources that can be accessed without regard for its physical components.

2.6.1 Overlay Networks

An overlay network is a virtual network of nodes and logical links that is built on top of an existing network with the purpose to implement a network service that is not available in the existing network.

The overlay network may sometimes change the properties of the underlying network. The overlay network provides:

- a service that is tailored towards the needs of a class of application or a particular higher-level service
- more efficient operation in a given networked environment
- additional feature – for example, multicast or secure communication.

Advantages:

- ✓ They enable new network services to be defined without requiring changes to the underlying network.
- ✓ 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.

Disadvantages:

- ✓ They introduce an extra level of indirection.
- ✓ They add to the complexity of network services.

Types of Overlay Networks

The overlay networks are classified based on the following criteria:

1. **Classification based on application:**
   - **Distributed hash tables:**

     A method for storing hash tables in geographically distributed locations in order to provide a failsafe lookupmechanism for distributed computing. The

overlay network offers a service that manages a mapping from keys to values across a potentially large number of nodes in a completely decentralized manner (similar to a standard hash table but in a networked environment).

- **Peer-to-peer file sharing:**

  Overlay structures that focus on constructing tailored addressing and routing mechanisms to support the cooperative discovery and use of files.

- **Content distribution networks:**

  Overlays that subsume a range of replication, caching and placement strategies to provide improved performance in terms of content delivery to web users; used for web acceleration and to offer the required real-time performance for video streaming.

2. **Classification based on network style:**

   - **Wireless ad hoc networks:** Network overlays that provide customized routing protocols for wireless ad hoc networks.

   - **Disruption-tolerant networks:** Overlays designed to operate in hostile environments that suffer significant node or link failure and potentially high delays

3. **Classification based on additional features:**

   - **Multicast:** Overlay networks provide access to multicast services where multicast routers are not available.

   - **Security:** Overlay networks that offer enhanced security over the underling IP network, including virtual private networks.

   - **Resilience:** Overlay networks that seek an order of magnitude improvement in robustness and availability of Internet paths

2.6.2 Skype –An overlay Network

Skype employs a partially decentralized architecture – a mix of the peer-to-peer and client-server architectures. The client-server system is used for authentication while the peer-to-peer system is used for IP telephony, relaying, indexing peers and file transfers. On top of this is the Skype overlay network which users interact with directly, an overlay network is a virtual network that is formed above and independent of the underlying Internet protocol network. Skype uses overlay networks to achieve the following:

1. It enables them to design and use their own proprietary protocols and application over the Internet. This enables Skype to encrypt all packet transmissions.

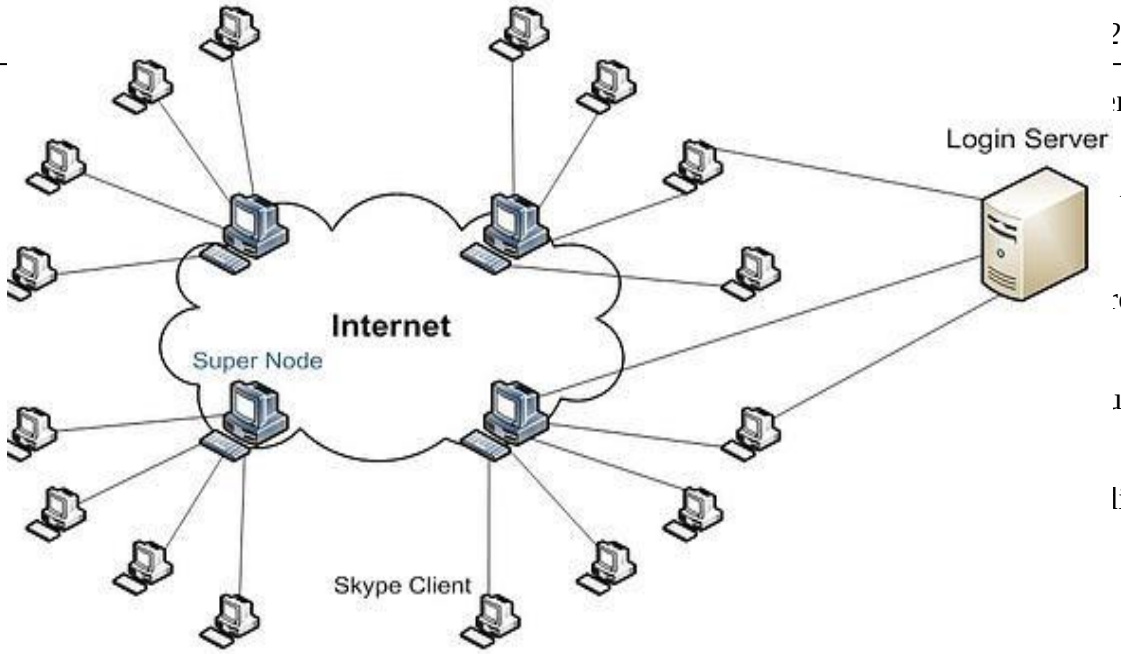


Fig 2.10: Skype Overlay Network

Skype architecture

- ✓ Skype is based on a peer-to-peer infrastructure consisting of ordinary users" machines called 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, reach ability and availability.

User connection

- ✓ Skype users are authenticated through a login server.
- ✓ They then make contact with a selected super node. To achieve this, each client maintains a cache of super node identities.
- ✓ 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

- ✓ Super nodes search 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.
- ✓ A user search typically takes between three and four seconds to complete for hosts that have a global IP address.

Voice connection

- ✓ Skype establishes a voice connection between the two parties using TCP for signaling all requests and terminations and either UDP or TCP for the streaming audio.

2.7 MESSAGE PASSING INTERFACE (MPI)

Message Passing Interface (MPI) is a standardized and portable message-passing system to function on a wide variety of parallel computers.

- MPI primarily addresses the message-passing parallel programming model: data is moved from the address space of one process to that of another process through cooperative operations on each process.
- The primary goal of the Message Passing Interface is to provide a widely used standard for writing message passing programs.
- The interface attempts to be: practical, portable, efficient and flexible.
- The basic principles of exchanging messages between two processes using send and receive operations is covered in MPI.

- There are two types of message passing:
  - **Synchronous message passing:** This is implemented by blocking send and receive calls.
  - **Asynchronous message passing:** This is implemented by a non-blocking form of send.

Originally, MPI was designed for distributed memory architectures.As architecture trends changed, shared memory SMPs were combined over networks creating hybrid distributed memory / shared memory systems. MPI runs on virtually any hardware platform like distributed memory, shared memory and in hybrid memory.

Send Operations	**Blocking Mode**	**Non Blocking Mode**
Generic	**MPI_Send**: the sender blocks until it is safe to return (i.e.) until the message is in transit or delivered and the sender‟s application buffer can therefore be reused.	**MPI_Isend:** the call returns immediately and the programmer is given a communication request handle, which can then be used to check the progress of the call via MPI_WaitorMPI_Test.
Synchronous	**MPI_Ssend:** the sender and receiver synchronize and the call only returns when the message has been delivered at the receiving end.	**MPI_Issend:** as with MPI_Isend,but with MPI_WaitandMPI_Testindicating whether the message has been delivered at the receive end.
Buffered	**MPI_Bsend:** the sender explicitly allocates an MPI buffer library and the call returns when the data is successfully copied into this buffer.	**MPI_Ibsend:** as with MPI_I send but with MPI_WaitandMPI_Testindicating whether the message has been copied into the sender‟s MPI buffer and hence is in transit.
Ready	**MPI_Rsend:** the call returns when the sender‟s application buffer can be reused but the programmer is also indicating to the library that the receiver is ready to receive the message, resulting in potential optimization of the underlying implementation.	**MPI_Irsend:** the effect is as with MPI_Isend, but as with MPI_Rsend, the programmer is indicating to the underlying implementation that the receiver is guaranteed to be ready to receive.

2.8 REMOTE INVOCATION

In a distributed object system, an object's data should be accessible only via its methods. The objects can be accessed via object references. The following terminologies are important in remote invocations:

- An **interface** provides a definition of the signatures of a set of methods without specifying their implementation.
- An **action** is initiated by an object invoking a method in another object.
- The **state** of an object consists of the values of its instance variables.

Remote invocations are method invocations for objects in different processes.

- **Remote objects** are objects that can receive remote invocation.

The core part of the distributed object model is:

- remote object reference;
- remote interface: specifies which methods can be invoked remotely;

2.8.1 Request reply protocols

The overview of request reply protocol is:

- The client sends a query to the server, and reads from the socket, usually in a blocking way, for the server response.
- The server processes the command and sends the response back to the client.

The Remote Procedure Call (RPC) is a common model of request reply protocol. There are two types of communication:

- **Synchronous:** The client process blocks until the reply arrives from the server. It is termed as reliable form of communication since the reply from the server is actually an acknowledgement to the client.
- **Asynchronous**: The clients can retrieve the replies from the server later.

UDP is preferred over TCP for the following reasons:

- Acknowledgements are redundant, since requests are followed by replies.
- Establishing a connection involves two extra pairs of messages in addition to the pair required for a request and a reply.
- Flow control is redundant for the majority of invocations, which pass only small arguments and results.

Request Reply protocol:

The following are the primitives in request reply protocols:

- **public byte[] doOperation (RemoteObjectRef o, intmethodId, byte[] arguments):**

  This method sends a request message to the remote object and returns the reply. The arguments specify the remote object, the method to be invoked and the arguments of that method.

- **public byte[] getRequest ():**

  This method acquires a client request via the server port.

- **public void sendReply (byte[] reply, InetAddressclientHost, intclientPort):**

  This method sends the reply message reply to the client at its Internet address and port.

Message identifiers

Each message have a unique message identifier for referencing it. A message identifier consists of two parts:

- request Id: Increasing sequence of integers assigned by the sending process. This makes the message unique to the sender.
- an identifier: for the sender process. This makes the message unique in the distributed system.

Failure model of the request-reply protocol

The following are the common failures in request reply protocol:

- Omission failures.
- Messages are not guaranteed to be delivered in sender order.
- Failure of processes.

Discarding duplicate request messages

The server may receive the same request message more than once. To avoid this, the protocol is designed to recognize successive messages from the same client with the same request identifier and to filter out duplicates.

Lost reply messages

If the server has already sent the reply to which it receives a duplicate request it will need to execute the operation again to obtain the result. This could be avoided if the server has stored the result of the original execution.

History

History is used to refer to a structure that contains a record of reply messages that have been transmitted by the server.

Each entry in the history has

- ✓ a request identifier
- ✓ a message
- ✓ client identifier

The main goal of history is to allow the server to retransmit replymessages when client processes request them. The main issue faced by the history is the size. As more communication takes place, more messages need to be saved in the history which incurs more cost.

Styles of exchange protocols

There are three exchange protocols:

- **request (R) protocol:** a single request message is sent by the client to the server
- **request-reply (RR) protocol:** used in client-server exchanges because it is based on the request-reply protocol. Special acknowledgement messages are not required.
- **request-reply-acknowledge reply (RRA) protocol:** It is based on request-reply-acknowledge reply. The Acknowledge reply message contains the requested from the reply message being acknowledged. This will enable the server to discard entries from its history. The arrival of a requested in an acknowledgement message will be interpreted as acknowledging the receipt of all reply messages with lower request Ids.

TCP streams to implement the request-reply protocol

It is difficult to determine the buffer size for the UDP datagrams. Also the fixed size of the datagrams imposes a limit on the message size which is not favorable. So TCP streams are preferred since they allow messages of any size to be transmitted. If the TCP protocol is used, it ensures that request and reply messages are delivered reliably thereby avoiding acknowledgments. The overhead due to TCP acknowledgement messages is reduced when a reply message follows soon after a request message.

HTTP: An Example Request Reply protocol

Hyper Text Transfer Protocol (HTTP) is a stateless request-response based communication protocol. It's used to send and receive data on the Web i.e., over the Internet. This protocol uses reliable TCP connections either for the transfer of data to and from clients (Web Browsers).

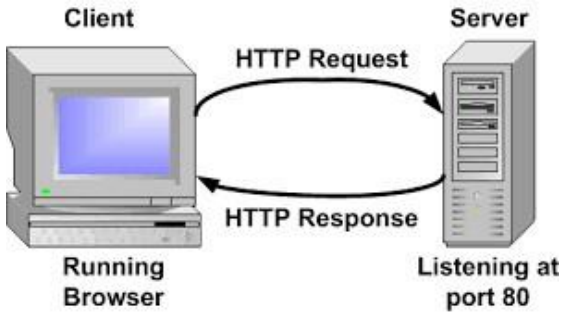


Fig 2.11: HTTP Request-Response protocol

HTTP specifies the messages involved in a request-reply exchange, the methods, arguments and results, and the rules for representing (marshalling) them in the messages. This protocol allows for content negotiation and password-style authentication

- **Content negotiation:** Clients‟ requests can include information as to what data representations they can accept.
- **Authentication:** Credentials and challenges are used to support password-style authentication.

The following are the connection establishment and closing steps:

- The client requests and the server accepts a connection at the default server port or at a port specified in the URL.
- The client sends a request message to the server.
- The server sends a reply message to the client.
- The connection is closed after the communication is over.

Connections in HTTP

There are two types of connections in HTTP: Persistent and non persistent

Differences between persistent and non persistent connections

Persistent Connection	Non persistent Connection
On the same TCP connection the server, parses request, responds and waits for new requests.	The server parses request, responds, and closes TCP connection.
It takes fewer RTTs to fetch the objects.	It takes 2 RTTs to fetch each object.
It has less slow start.	Each object transfer suffers from slow start

HTTP Request Message

Method	URL	HTTP version	Headers	Message body
GET	http://www.abc.ac.ind/data.html	HTTP/1.1		

Fig 2.12: HTTP Request Message

The following are the parts of HTTP request frame :

- **Request Method:** The request **method** indicates the method to be performed on the resource identified by the given **Request-URL**. The method is case-sensitive and should always be mentioned in uppercase.
- **Request-URL:** This Uniform Resource Identifier and identifies the resource upon which to apply the request.
- **Request Header Fields:** The request-header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers.
- **HTTP Version:** This specifies the version of HTTP used.

HTTP Methods

The following are the HTTP methods:

- **GET:** The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
- **HEAD:** Same as GET, but it transfers the status line and the header section only.
- **POST:** A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
- **PUT:** Replaces all the current representations of the target resource with the uploaded content.
- **DELETE:** Removes all the current representations of the target resource given by URI.
- **CONNECT:** Establishes a tunnel to the server identified by a given URI.
- **OPTIONS:** Describe the communication options for the target resource.
- **TRACE:** Performs a message loop back test along with the path to the target resource.

HTTP Response Message

HTTP Version	Status Code	Reason	Headers	Message body
HTTP/1.1	2xx	OK		Data

Fig 2.13: HTTP Response Message

A response message consists of the protocol version followed by a numeric status code and its associated textual phrase.

- HTTP Version: A server supporting HTTP version 1.1 will return the following version information: HTTP-Version = HTTP/1.1
- Status Code: The Status-Code element is a 3-digit integer where first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. There are 5 values for the first digit:
  - ✓ 1xx- Informational :It means the request was received and the process is continuing.
  - ✓ 2xx- Success: It means the action was successfully received, understood, and accepted.
  - ✓ 3xx-Redirection: It means further action must be taken in order to complete the request.
  - ✓ 4xx- Client Error: It means the request contains incorrect syntax or cannot be fulfilled.
  - ✓ 5xx- Server error: It means the server failed to fulfill an apparently valid request.

HTTP status codes are extensible and HTTP applications are not required to understand the meaning of all registered status codes.

- **Response Header Fields:** The response-header fields allow the server to pass additional information about the response which cannot be placed in the Status-Line. These header fields give information about the server and about further access to the resource identified by the Request-URI.

2.8.2 Remote Procedure Call (RPC)

RPC is a powerful technique for constructing distributed, client-server based applications. In RPC, the procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them. By using RPC, programmers of distributed applications avoid the details of the interface with the network. RPC makes the client/server model of computing more powerful and easier to program.

Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details.

Design issues in RPC

The following are the design issues in RPC:

- **Programming with interfaces**
  - Communication between modules in a program can be done by procedure calls between modules or by direct access to the variables in another module.
  - To control the interactions between modules, an explicit interface is defined for each module that specifies the procedures and the variables that can be accessed from other modules.
  - The interface hides the programming details of one module from the other.
  - The interaction between the modules can happen even after the modification of the modules but without modifying the interfaces.
- **Interfaces in distributed systems**
  - Each server in the client server model provides a set of procedures that are available for use by clients.
  - The separation between interface and implementation offers the following advantages:
    - ✓ The programmers need not be aware of implementation details of each modules.
    - ✓ In the perspective of distributed systems, the programmers need to know the programming language or underlying platform used to implement the service.
    - ✓ It provides support for software evolution in which implementations can change as long as long as the interface remains the same. But the interface can also change as long as it remains compatible with the original.
- **Interface definition languages:**
  - An RPC mechanism can be integrated with a particular programming language if it contains notations for defining interfaces, allowing input and output parameters to be mapped onto the language‟s normal use of parameters.

- But programs can also be written in a variety of languages to access them remotely.
- In such cases Interface definition languages (IDL) are used which contains procedures implemented indifferent languages to invoke one another.

❖ **RPC call semantics**

- In request-reply protocols doOperation can be implemented in different ways to provide different delivery guarantees:
  - ✓ 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 above choices lead to a variety of possible semantics as given below:

a) **Maybe semantics:** Here, the remote procedure call may be executed once or not at all. Maybe semantics arises when no fault-tolerance measures are applied and can suffer from the following types of failure:

- ✓ omission failures if the request or result message is lost
- ✓ crash failures.

May be semantics is useful only for applications in which occasional failed calls are acceptable.

b) **At-least-once semantics:** Here 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 suffer from the following types of failure:

- ✓ crash failures when the server containing the remote procedure fails
- ✓ arbitrary failures

c) **At-most-once semantics:** Here**,** 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.

❖ **Transparency**

- The remote procedure calls are much like local procedure calls as possible, with no distinction in syntax between a local and a remote procedure call.

- All the necessary calls to marshalling and message-passing procedures were hidden from the programmer making the call.
- Although request messages are retransmitted after a timeout, this is transparent to the caller to make the semantics of remote procedure calls like that of local procedure calls
- The usage of middleware can also offer additional levels of transparency to RPC.
- But RPC‟s are more vulnerable to failure than local calls since they involve a network, another computer and another process.
- So the clients making remote calls must be able to recover from such situations.
- The latency of a remote procedure call is several orders of magnitude greater than that of a local one.

Implementation of RPC

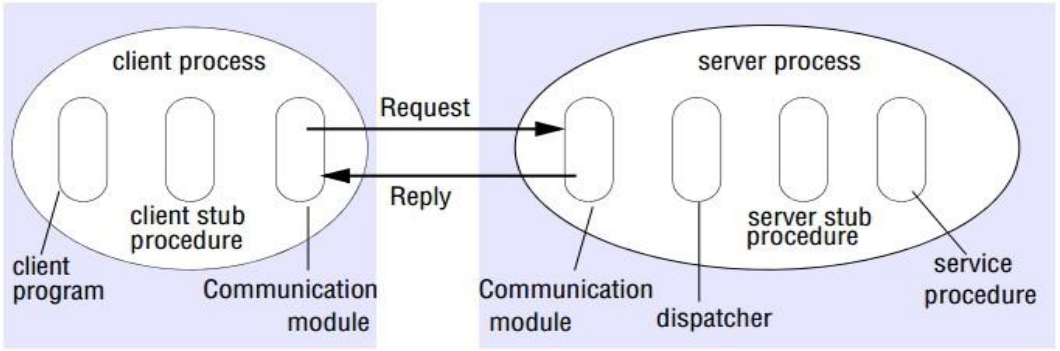


Fig 2.14: Client and Server stub in RPC

- The client that accesses a service includes one stub procedure for each procedure in the service interface.

A stub is a piece of code that is used to convert parameters during a remote procedure call (RPC). An RPC allows a client computer to remotely call procedures on a server computer.

- The parameters used in a function call have to be converted because the client and server computers use different address spaces.
- Stubs perform this conversion so that the remote server computer perceives the RPC as a local function call.
- The client stub routine performs the following functions:

- ✓ **Marshals arguments:** The client stub packages input arguments into a form that can be transmitted to the server.
- ✓ Calls the client run-time library to transmit arguments to the remote address space and invokes the remote procedure in the server address space.
- ✓ **Unmarshals output argument:** The client stub unpacks output arguments and returns to the caller.

- The server stub procedure unmarshals the arguments in the request message, calls the corresponding service procedure and marshals the return values for the reply message.
- The client and server stub procedures and the dispatcher are automatically generated by an interface compiler from the interface definition of the service.

2.8.3 Remote Method Invocation (RMI)

RMI is a set of protocols being developed by Sun's JavaSoft division that enables Java objects to communicate remotely with other Java objects.

RMI is closely related to RPC but extended into the world of distributed objects. RMI is a true distributed computing application interface specially designed for Java, written to provide easy access to objects existing on remote virtual machines

Differences between RMI and RPC

RMI	RPC
RMI uses an object oriented paradigm where the user needs to know the object and the method of the object he needs to invoke.	RPC is not object oriented and does not deal with objects. Rather, it calls specific subroutines that are already established
With RPC looks like a local call. RPC handles the complexities involved with passing the call from the local to the remote computer.	RMI handles the complexities of passing along the invocation from the local to the remote computer. But instead of passing a procedural call, RMI passes a reference to the object and the method that is being called.

The commonalities between RMI and RPC are as follows:

- ✓ They both support programming with interfaces.
- ✓ They are constructed on top of request-reply protocols.
- ✓ They both offer a similar level of transparency.

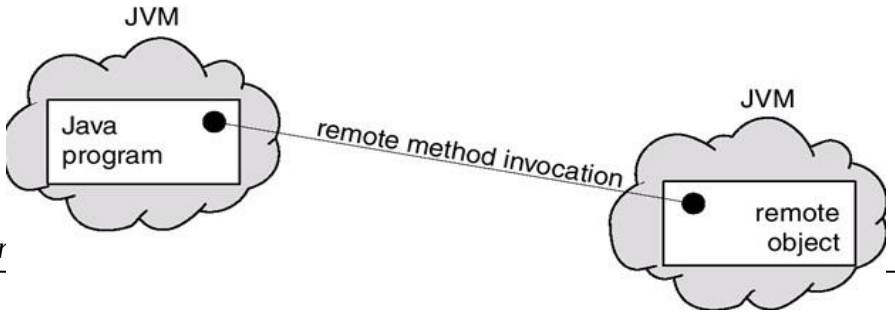


Fig 2.15: RMI in JVM

Features of RMI

- ✓ In RMI, the remote objects are treated similarly to local objects.
- ✓ RMI provide access to objects existing on remote virtual machines.
- ✓ RMI handles marshalling, transportation, and garbage collection of the remote objects.
- ✓ RMI was incorporated as a part of the JDK with version 1.1

Limitations of RMI

- ✓ RMI is not language independent. It is limited only to Java.
- ✓ Interfaces to remote objects are defined using ordinary Java interfaces not with special IDLs.

Design issues of RMI

The RMI evolved from simple object model to distributed objects model.

a) Object Oriented Model

An object communicates with other objects by invoking their methods by arguments and return values. Objects can encapsulate their dataand the code of their methods. In a distributed object system, an object‟s data should be accessible only through its methods.

⇒ **Object references:**

- Objects are accessed through object references.
- Object references could be assigned to variables, passed as arguments and returned as results of methods.

⇒ **Interfaces:**

- An interface provides implementation independent, definition of the signatures of a set of methods.

- An interface is not a class. A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements.
- An interface can contain any number of methods.
- An interface is written in a file with a .java extension, with the name of the interface matching the name of the file.
- The byte code of an interface appears in a .class file.
- Interfaces do not have constructors.

Differences between Interfaces and Class

Interfaces	**Class**
Interfaces cannot be instantiated.	Classes can be instantiated.
They do not contain constructors.	They can have constructors.
They cannot have instance fields (i.e.) all the fields in an interface must be declared both static and final.	No constraints over the fields.
The interface can contain only abstract method.	They can contain method implementation.
Interfaces are implemented by a class not extended.	Classes are extended by other classes.

⇒ **Actions :**

- Action is initiated by an object invoking a method in another object with or without arguments.
- The receiver executes the appropriate method and thenreturns control to the invoking object.
- The following are the effects of actions:
  - The state of the receiver may be changed.
  - A new object may be instantiated.
  - Further invocations on methods in other objects may take place.

⇒ **Exceptions:**

- Exceptions are run time errors.
- The programmers must write code that could handle all possible unusual or erroneous cases.

⇒ **Garbage collection:**

– This is a method in Java to free the space occupied by obsolete objects.

b) **Distributed objects**

⇒ Distributed object systems may adopt the client-server architecture where the objects are managed by servers and their clients invoke their methods using remote method invocation.

⇒ In RMI, the client‟s request to invoke a method of an object is sent in a message to the server managing the object.

⇒ The invocation is carried out by executing a method of the object at the server and the result is returned to the client in another message.

⇒ Objects with different data formats may be used at different sites.

c) **The distributed object model**

In this model, each process contains a collection of objects, some of which can receive both local and remote invocations, whereas the other objects can receive only local invocations. The core concepts of distributed object model:

- **Remote object references:** Other objects can invoke the methods of a remote object if they have access to its remote object reference.
- **Remote interfaces:** Every remote object has a remote interface that specifies which of its methods can be invoked remotely.
- **Remote object references:** The notion of object reference is extended to allow any object that can receive an RMI to have a remote object reference.

A remote object reference is an identifier that can be used throughout a distributed system to refer to a particular unique remote object.

The remote object to receive a remote method invocation is specified by the invoker as a remote object reference. The remote object references may be passed as arguments and results of remote method invocations.

- **Remote interfaces**: The class of a remote object implements the methods of its remote interface. Objects in other processes can invoke only the methods that belong to its remote interface.

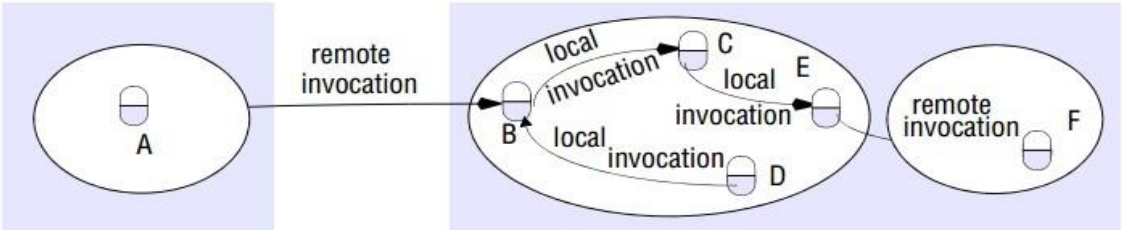


Fig 2.16: Remote and local invocations

Differences between remote and local objects

Remote Objects	**Local Objects**
Creating a remote object requires creating a stub and a skeleton, which will cost more time cycles.	They consume less time cycles.
Marshalling and unmarshalling is required.	Marshalling and unmarshalling is not required.
Longer time is taken by the remote object to return to the calling program.	Comparatively shorter return time.

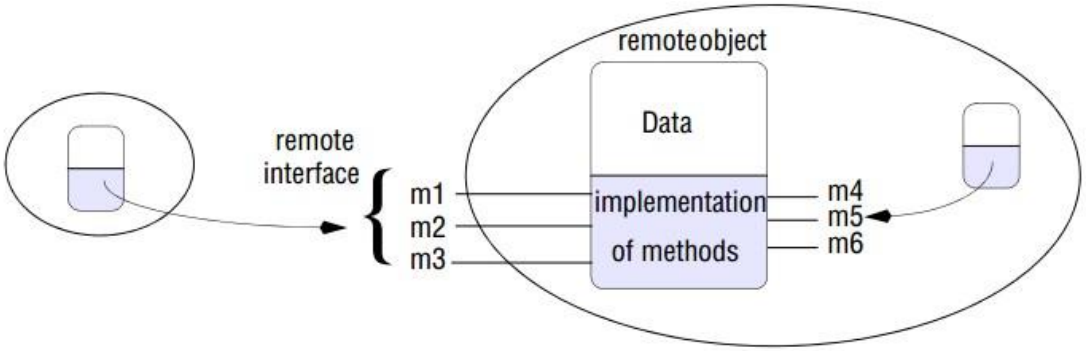


Fig 2.17: Remote Object and remote interfaces

Garbage collection in a distributed-object system:

Distributed garbage collection is usually based on reference counting. The total number of references to the object is maintained by a **reference count field**. The advantage of using reference counts is that garbage is easily identified. When it becomes necessary to reclaim the storage from unused objects, the garbage collector needs only to examine the reference count fields of all the objects that have been created by the program. If the reference count is zero, the object is garbage.

Exceptions:

Any remote invocation may fail because of exceptions. So RMI should include exception handling as a part of their implementation.

Implementation of RMI

Implementation of a RMI requires construction if many modules and the supporting interfaces.

a) Modules:

The two common modules used are communication module and remote reference module.

Communication module:

- ✓ This module implements request-reply protocol.
- ✓ This module specifies the message type, its requestId and the remote reference of the object to be invoked.
- ✓ This specifies only the communication semantics.
- ✓ The communication module in the server selects the dispatcher for the class of the object to be invoked, passing on its local reference, which it gets from the remote reference module in return for the remote object identifier in the request message.

Remote reference module:

- ✓ This module is responsible for translation between local and remote object references and for creating remote object references.
- ✓ This module has a remote object table specific for each process that records the mapping between local object references and remote object references.
- ✓ The entries of the table includes:
  - the remote objects held by the process.
  - local proxy
- ✓ The actions of them remote reference module are as follows:
  - When a remote object is to be passed as an argument for the first time, the remote reference module creates a remote object reference and is added to the table.
  - When a remote object reference arrives in a request or reply message the remote object table is referred for the corresponding local object reference.
  - If the remote object reference is not in the table, the RMI software creates a new proxy and asks the remote reference module to add it to the table.
  - This module is called by components of the RMI software during marshalling and unmarshalling remote object references.

b) Supporting Components

Servants

- ✓ A servant is an instance of a class that provides the body of a remote object.
- ✓ The responsibility of the servant is to handle the remote requests passed on by the corresponding skeleton.
- ✓ Servants live within a server process.
- ✓ They are created when remote objects are instantiated and remain in use until they are no longer needed and finally servants are garbage collected or deleted.

The RMI software

- ✓ This consists of the application-level objects and the communication and remote reference modules. The RMI software has the following components:
  - **Proxy:** The class of the proxy contains a method corresponding to each method in the remote interface which marshalls arguments and unmarshalls results for that method. It hides the details of the remote object reference. Each method of the proxy marshals a reference to the target object using a request message (operationId, arguments). When the reply is received the proxy, unmarshals it and returns the results to the invoker.
  - **Dispatcher:** Server has a dispatcher & a skeleton for each class of remote object. The dispatcher receives the incoming message, and uses its method info to pass it to the right method in the skeleton.
  - **Skeleton:** Skeleton implements methods of the remote interface to unmarshall arguments and invoke the corresponding method in the servant. It then marshalls the result (or any exceptions thrown) into a reply message to the proxy.

Generation of the classes for proxies, dispatchers and skeletons

- ✓ The classes for the proxy, dispatcher and skeleton used in RMI are generated automatically by an interface compiler.
- ✓ For Java RMI, the set of methods offered by a remote object is defined as a **Java interface.**

Dynamic invocation:

- ✓ This is an alternative to proxies.
- ✓ The proxy just described (i.e.) its class is generated from an interface definition and then compiled into the client code.

- ✓ If a client program receives a remote reference to an object whose remote interface was not available at compile time. The method could not be invoked through proxies.
- ✓ Dynamic invocation is used in this case which gives the client access to a generic representation of a remote invocation, which is a part of the infrastructure for RMI.
- ✓ The client will supply the remote object reference, the name of the method and the arguments to doOperation and then wait to receive the results.

Dynamic skeletons:

The Dynamic Skeleton Interface (or DSI) allows applications to provide implementations of the operations on CORBA objects without static knowledge of the object's interface. It is the server-side equivalent of the Dynamic Invocation Interface.

Server programs

The server program contains

- ✓ the classes for the dispatchers and skeletons, with the implementations of the classes of all of the servants that it supports.
- ✓ an initialization section which is responsible for creating and initializing at least one of the servants to be hosted by the server.

Client programs

The client program will contain

- ➢ the classes of the proxies for all of the remote objects that it will invoke.

Factory methods

- ✓ Factory methods are static methods that return an instance of the native class.
- ✓ Factory method is used to create different object from factory often refereed as Item and it encapsulate the creation code.
- ✓ So instead of having object creation code on client side we encapsulate inside Factory method in Java.
- ✓ The term factory method is sometimes used to refer to a method that creates servants, and a factory object is an object with factory methods.
- ✓ Any remote object that needs to be able to create new remote objects on demand for clients must provide methods in its remote interface for this purpose. Such methods are called factory methods, although they are really just normal methods.

Binder

- ✓ A binder in a distributed system is a separate service that maintains a table containing mappings from textual names to remote object references.
- ✓ It is used by servers to register their remote objects by name and by clients to look them up.

Server threads

- ✓ Whenever an object executes a remote invocation, that execution may lead to further invocations of methods in other remote objects, which may take sometime to return.
- ✓ To avoid this, servers generally allocate a separate thread for the execution of each remote invocation.

Activation of remote objects

- ✓ Activation of servers to support remote objects is done by a service called **Inetd**.
- ✓ Certain applications demand the objects to persist for longer times.
- ✓ When the objects are maintained for longer period it is actually waste of resources.
- ✓ To avoid this the servers could be started on demand as and when needed through Inetd.
- ✓ Processes that start server processes to host remote objects are called **activators.**
- ✓ Based on this remote objects can be described as active or passive.
  - ➢ An object is said to be **active** when it is available for invocation.
  - ➢ An object is called **passive** if is currently inactive but can be made active.
- ✓ A passive object consists of two parts:
  - ➢ implementation of its methods
  - ➢ its state in the marshalled form.
- ✓ When activating a passive object, an active object is created from the corresponding passive object by creating a new instance of its class and initializing its instance variables from the stored state.
- ✓ Passive objects can be activated on demand.

Responsibilities of an activator:

- ✓ Registering passive objects that are available for activation. This is done by mapping the names of servers and passive objects.

- ✓ Starting named server processes and activating remote objects in them.
- ✓ Tracking the locations of the servers for remote objects that it has alreadyactivated.
- ✓ Java RMI provides the ability to make some remote objects activatable [java.sun.comIX].

Persistent object stores

An object that is guaranteed to live between activations of processes is called a persistent object.

- ✓ Persistent objects are managed by persistent object stores, which store their state in a marshalled form on disk.
- ✓ The objects are activated when their methods are invoked by other objects.
- ✓ When activating the object, the invoker will not know whether an object is in main memory or in a disk.
- ✓ The persistent objects that are not needed are maintained in the object store (active or passive state).
- ✓ The decision of making an active object to passive is done through the following ways:
  - ➢ in response to a request in the program that activated the objects
  - ➢ at the end of a transaction
  - ➢ when the program exits.
- ✓ There are two approaches to determine whether an object is persistent or not:
  - ➢ The persistent object store maintains some persistent roots, and any object that is reachable from a persistent root is defined to be persistent.
  - ➢ The persistent object store provides some classes. If the object belong to their subclasses, then it is persistent.

Object location

- ✓ Remote object reference can be done based on the Internet address and port number of the process that created the remote object.
- ✓ Some remote objects will exist in different processes, on different computers, throughout their lifetime.
- ✓ In this case, a remote object reference cannot act as an address for the objects.

- ✓ So clients making invocations require both a remote object reference and an address to which to send invocations.
- ✓ A **location service** helps clients to locate remote objects from their remote object references.
- ✓ It acts as a database that maps remote object references to their probable current locations.
- ✓ Alternatively a **cache/broadcast scheme** could also be used.
- ✓ Here, a member of a location service on each computer holds a small cache of remote object reference to-location mappings.
- ✓ If a remote object reference is in the cache, that address is tried for the invocation and will fail if the object has moved.
- ✓ To locate an object that has moved or whose location is not in the cache, the system broadcasts a request.

Distributed garbage collection

- ✓ It is always desirable to automatically delete the remote objects that are no longer referenced by any client.
- ✓ The memory held by those objects will be recovered.
- ✓ RMI uses a **reference-counting garbage collection** algorithm for automatic garbage collection.
- ✓ In this algorithm, the RMI runtime keeps track of all live references within each Java virtual machine.
- ✓ When a live reference enters a Java virtual machine, its reference count is incremented.
- ✓ The first reference to an object sends a referenced message to the server for the object.
- ✓ As live references are found to be unreferenced in the local virtual machine, the count is decremented.
- ✓ When the last reference has been discarded, an unreferenced message is sent to the server.
- ✓ The Java distributed garbage collection algorithm can tolerate the failure of client processes.
- ✓ To achieve this, servers lease their objects to clients for a limited period of time.

- ✓ The lease period starts when the client makes first referenceinvocation to the server.
- ✓ It ends either when the time has expired or when the client makes a last reference invocation to the server.
- ✓ The information stored by the server (i.e.) the leasecontains the identifier of the client‟s virtual machine and the period of the lease.
- ✓ Clients are responsible for requesting the server to renew their leases before they expire.

Leases in Jini

- ✓ Jini provides a mechanism for locating services on the network that conform to a particular (Java) interface, or that have certain attributes associated with them.
- ✓ Once a service is located, the client can download an implementation of that interface, which it then uses to communicate with the service.
- ✓ The Jini distributed system includes a specification for leases that can be used in a variety of situations when one object offers a resource to another object.
- ✓ The granting of the use of a resource for a period of time is called a **lease**.
- ✓ The object offering the resource will maintain it until the time in the lease expires.
- ✓ The resource users are responsible for requesting their renewal when they expire.
- ✓ The period of a lease may be negotiated between the grantor and the recipient in Jini.
- ✓ In Jini, an object representing a lease implements the Lease interface.
- ✓ It contains information about the period of the lease and methods enabling the lease to be renewed or cancelled.

2.9 CASE STUDY: JAVA RMI

In this case study we use shared white board example. This program that allows a group of users to share a common view of a drawing surface containing graphical objects, such as rectangles, lines and circles, each of which has been drawn by one of the users.

Remote interfaces in Java RMI

- ✓ Remote interfaces are defined by extending an interface called Remote in the java.rmipackage.
- ✓ The methods must throwRemoteException,contain an GraphicalObject which implements theSerializableinterface.

Java Remote interfaces Shape and ShapeList

```
import java.rmi.*;
import java.util.Vector;
public interface Shape extends Remote
 {
intgetVersion() throws RemoteException;
GraphicalObjectgetAllState() throws RemoteException;
}
public interface ShapeList extends Remote
 {
   Shape newShape(GraphicalObject g) throws RemoteException;
   Vector allShapes() throws RemoteException;
intgetVersion() throws RemoteException;
}
```

Parameter and result passing

The parameters of a method are input parameters and the result of a method is a single output parameter. Any object that implements the Serializable interface can be passed as an argument or result in Java RMI. All primitive types and remote objects are serializable.

- **Passing remote objects:**
  - ✓ When the type of a parameter or result value is defined as a remote interface, the corresponding argument or result is always passed as a remote object reference.
  - ✓ When a remote object reference is received, it can be used to make RMI calls on the remote object to which it refers.
- **Passing non-remote objects:**
  - ✓ All serializable non-remote objects are copied and passed by value.
  - ✓ When an object is passed by value, a new object is created in the receiver‟s process.
  - ✓ The methods of this new object can be invoked locally, possibly causing its state to differ from the state of the original object in the sender‟s process.
  - ✓ In this example, the client uses the method new Shape that passes an instance of GraphicalObject to the server.
  - ✓ The server creates a remote object of type Shape with the state of the GraphicalObject and returns a remote object reference to it.

- ✓ The arguments and return values in a remote invocation are used with the following modifications:

    1. Whenever an object that implements the Remote interface is serialized, it is replaced by its remote object reference, which contains the name of its (the remote object‟s) class.

    2. When any object is serialized, its class information is annotated with the location of the class (as a URL), enabling the class to be downloaded by the receiver.

Downloading of classes

- ✓ In Java classes can be downloaded from one virtual machine to another.
- ✓ If the recipient does not already possess the class of an object passed by value, its code is downloaded automatically.
- ✓ If the recipient of a remote object reference does not already possess the class for a proxy, its code is downloaded automatically.
- ✓ The advantages are:

    1. There is no need for every user to keep the same set of classes in their working environment.

    2. Both client and server programs can make transparent use of instances of new classes whenever they are added.

RMIregistry

- ✓ The RMIregistry is the binder for Java RMI.
- ✓ It maintains a table mapping URL-style names to references to remote objects.
- ✓ The Naming class is used to access the RMIregistry.
- ✓ The methods of the naming class is of the general form:

    //computerName:port/objectName
    where computerName and port refer to the location of the RMIregistry.

- ✓ The LocateRegistryclass , which is in java.rmi.registry, is used to discover the names.
- ✓ The getRegistry()returns an object of typeRegistry representing the remote binding service:
    public static Registry getRegistry() throws RemoteException

The following table gives a description of the methods in the Naming class:

Method	**Description**
void rebind (String name, Remote obj)	This method is used by a server to register the identifier of a remote object by name.
void bind (String name, Remote obj)	This method is also used to register a remote object by name, but if the name is already bound to a remote object reference an exception is thrown.
void unbind (String name, Remote obj)	This method removes a binding.
Remote lookup(String name)String [] list()	This method returns an array of Strings containing the names bound in the registry .This method is used by clients to look up a remote object by name.

Server Program

```
import java.util.Vector;
public class ShapeListServant implements ShapeList
{
    private Vector theList; // contains the list of Shapes
    private int version;
    public ShapeListServant(){...}
    public Shape newShape(GraphicalObject g)
    {
        version++;
        Shape s = new ShapeServant( g, version);
theList.addElement(s);
        return s;
    }
public Vector allShapes(){...}
public intgetVersion() { ... }
}
```

Client Program

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.Vector;
public class ShapeListClient
{
    public static void main(String args[])
   {
System.setSecurityManager(new RMISecurityManager());
ShapeListaShapeList = null;
            try
           {
aShapeList = (ShapeList) Naming.lookup("//project.ShapeList");
                  Vector sList = aShapeList.allShapes();
            }
        catch(RemoteException e)
       {
System.out.println(e.getMessage());
        }
       catch(Exception e)
       {
System.out.println("Client: " + e.getMessage());
      }
  }
}
```

Callbacks

Callback refers to a server‟s action of notifying clients about an event. Callbacks can be implemented in RMI as follows:

- ✓ The client creates a remote object that implements an interface that contains a method for the server to call.
- ✓ The server provides an operation allowing interested clients to inform it of the remote object references of their callback objects.
- ✓ Whenever an event of interest occurs, the server calls the interested clients.

The disadvantages of callbacks:

- The performance of the server may be degraded by the constant polling.
- Clients cannot notify users of updates in a timely manner.

Design and implementation of Java RMI

Use of reflection

- ✓ Reflection is used to pass information in request messages about the method to be invoked.
- ✓ This is achieved with the help of the class Method in the reflection package.
- ✓ Each instance of Method represents the characteristics of a particular method, including its class and the types of its arguments, return value and exceptions.
- ✓ An instance of Method can be invoked on an object of a suitable class by means of its invoke method.
- ✓ The invoke method requires two arguments: the first specifies the object to receive the invocation and the second is an array of Object containing the arguments. The result is returned as type Object.
- ✓ The proxy has to marshal information about a method and its arguments into the request message with array of Objects as its arguments and then marshals that array.
- ✓ The dispatcher ; arguments in the array of Objects from th
- ✓ The dispatcher method, supplying the target and the array od has been executed, the dispatcher mars e reply message.

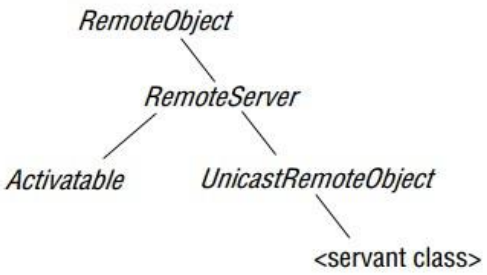


Fig 2.13: Classes in Java RMI

Java classes supporting RMI

- Every servent need to extend UnicastRemoteObject.

- The class UnicastRemoteObject extends an abstract class called RemoteServer, which provides abstract versions of the methods required by remote servers.
- Activatable class is available for providing activatable objects.
- The class RemoteServer is a subclass of RemoteObject that has an instance variable holding the remote object reference and provides the following methods:
  - equals: Compares remote object references.
  - toString: Returns the contents of the remote object reference as a String
  - readObject, writeObject: These methods deserialize/serialize remote objects

2.10 GROUP COMMUNICATION

Group communication is a service where a message is sent to a group and then this message is delivered to all members of the group.

Programming Model

A group has group members (processes) who can join or leave the group. This is done through multicast communication. There are three communication modes:

- **Unicast:** Process to process communication
- **Broadcast:** Communication to all the process
- **Multicast:** Communication only to a group of processes

Process Groups and Objects Groups

- ✓ An object group is a collection of objects that process the same set of invocations concurrently.
- ✓ Client objects invoke operations on a single, local object, which acts as a proxy for the group.
- ✓ The proxy uses a group communication system to send the invocations to the members of the object group.
- ✓ Object parameters and results are marshalled as in RMI and the associated calls are dispatched automatically to the right destination objects/methods.

Types of group communication

- **Closed and open groups:** A group is said to be closed if only members of the group may multicast to it A process in a closed group delivers to itself any message that it multicasts to the group.

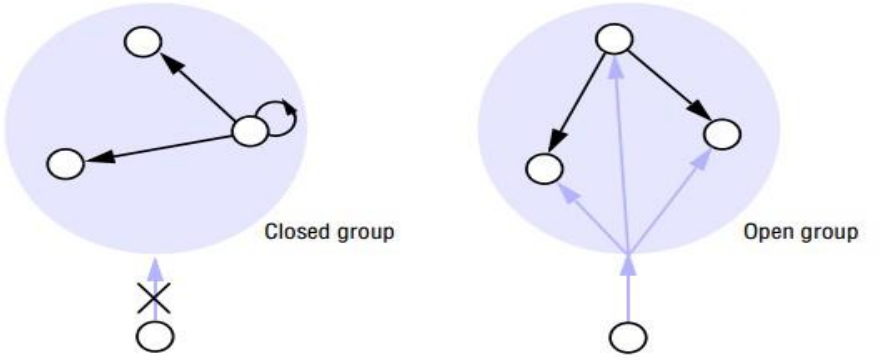


Fig 2.15: Open and Closed group communication

- **Overlapping and non-overlapping groups:** In overlapping groups, entities may be members of multiple groups, and non-overlapping groups imply that membership does not overlap to at most one.
- **Synchronous and asynchronous systems**: Group communication is possible in both environments.

Reliability and ordering in multicast

- ✓ In group communication, all members of a group must receive copies of the messages sent to the group, with delivery guarantees.
  - ➢ Reliable multicast operation is based on:
  - ➢ Integrity: delivering the message correctly only once
  - ➢ Validity: guaranteeing that a message sent will eventually be delivered.
- ✓ Group communication demands relative ordering of messages delivered to multiple destinations.
- ✓ The following are the types of message ordering:
  - ➢ **FIFO ordering:** If a process sends one message before another, it will be delivered in this order at all processes in the group.
  - ➢ **Causal ordering:** 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.

Group membership management

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.
- **Failure detection:** The service monitors the group members in case of crash and unreachability. The detector marks processes as Suspected or Unsuspected. The process is excluded from membership if it is not reachable or crashed.
- **Notifying members of group membership changes:** The service notifies the group‟s members when a process is added, or when a process is excluded.
- **Performing group address expansion:** When a process multicasts a message, it supplies the group identifier rather than a list of processes in the group.

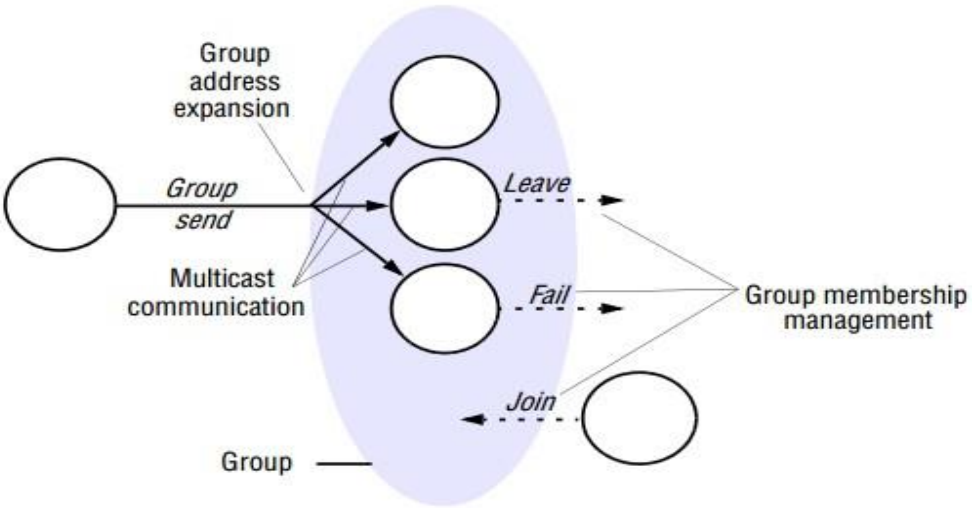


Fig 2.15: Group membership management

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. The toolkit supports a variety of reliability and ordering guarantees.

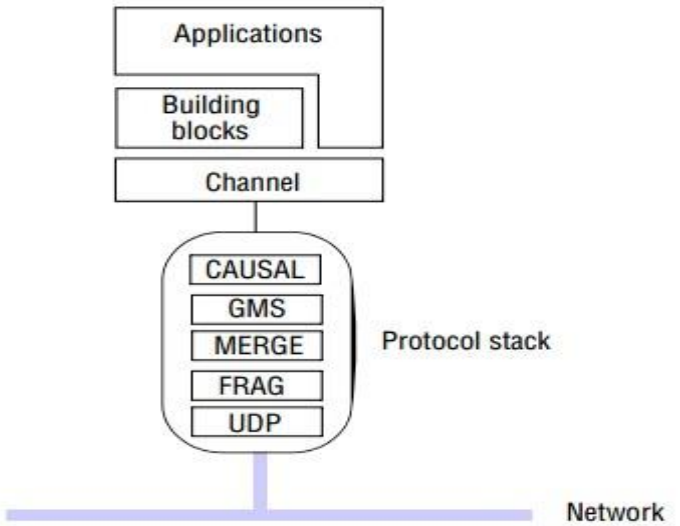


Fig 2.16: Architecture of JGroups

The following are the important components in the architecture:

- **Channels**
  - ✓ They represent the most primitive interface for application developers, offering the core functions of joining, leaving, sending and receiving.
  - ✓ A process interacts with a group through a channel object, which acts as a handle onto a group.
  - ✓ The following operations are allowed in channel:
    - Connect: binds that handle to a particular named group.If the named group does not exist, it is implicitly created at the time of the first connect.
    - Disconnect: To leave the group, the process executes the corresponding disconnect operation.
    - Close: will render the channel unusable.
    - getView: returns the current view defined in terms of the current member list
    - getState: returns the historical application state associated with the group
- **Building blocks**
  - ✓ This offers higher-level abstractions, building on the underlying service offered by channels.

- ✓ Some of the building blocks are:
  - **MessageDispatcher**
    - 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.
  - **RpcDispatcher**
    - This takes a specific method and invokes this method on all objects associated with a group.
  - **NotificationBus**
    - This 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.

➢ **Protocol stack**

- ✓ This provides the underlying communication protocol, constructed as a stack of protocol layers.
- ✓ Here, 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);
- ✓ Each layer process the message, including modifying its contents, adding a header or indeed dropping or reordering the message.
- ✓ This shows a protocol that consists of five layers:
  - UDP: This is the transport layer that utilizes IP multicast and UDP datagrams specifically for point-to-point communication. For larger-scale systems operating over wide area networks, a TCP layer may be preferred.
  - FRAG: This implements message packetization and the maximum packet size is 8,192 bytes by default.
  - MERGE: This deal with unexpected network partitioning and the subsequent merging of subgroups after the partition.
  - GMS: This implements a group membership protocol to maintain consistent views of membership across the group
  - CAUSAL: This implements causal ordering.

2.11 PUBLISH -SUBSCRIBER SYSTEMS

- ✓ Publish–subscribe is a messaging pattern where senders of messages, called **publishers,** do not program the messages to be sent directly to specific receivers, called **subscribers**.
- ✓ Instead, published messages are characterized into classes. The subscribers express interest in one or more classes, and only receive messages that are of interest.
- ✓ The task of the publish subscribe system is to match subscriptions against published events and ensure the correct delivery of event notifications.

Characteristics of publish-subscribe systems

- ➢ **Heterogeneity:** In distributed environment, the different types of components are made to interoperate with each other. To maintain integrity an interface is designed for receiving and dealing with the resultant notifications.
- ➢ **Asynchronicity:** Notifications are sent asynchronously by event-generating publishers to all the subscribers that have expressed an interest. This avoids the need for synchronizing.

The programming model

- ✓ Publishers disseminate an event e through a publish(e) operation and subscribers express an interest in a set of events through subscribe(f).
- ✓ This refers to a filter or a pattern defined over the set of all possible events.
- ✓ Subscribers can later revoke this interest through a corresponding unsubscribe(f) operation.
- ✓ The events are delivered to the subscriber using a notify(e) operation.
- ✓ The advertise(f) allows the publishers to have the option of declaring the nature of future events.

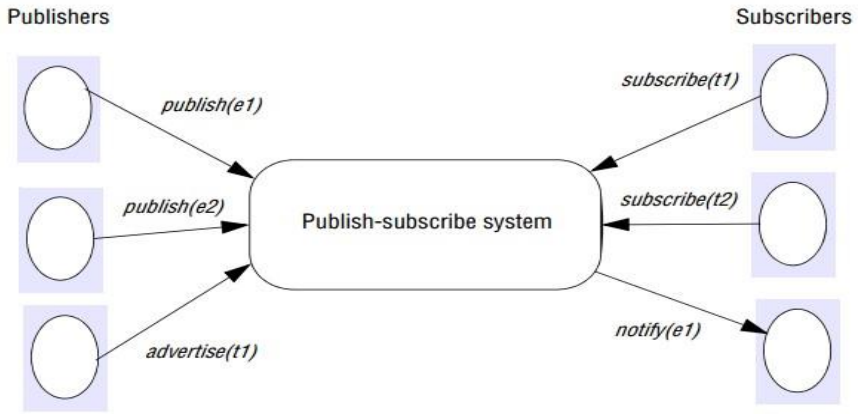


Fig 2.17: Publish Subscriber Model

The following are the subscriber or filter models:

- **Channel-based:**
  - ✓ The publishers publish events to named channels and subscribers then subscribe to one of these named channels to receive all events sent to that channel.
- **Topic-based (subject-based):**
  - ✓ Each notification from the publisher is expressed in terms of a number of fields, with one field denoting the topic.
  - ✓ Subscriptions are then defined in terms of the topic of interest.
- **Content-based:**
  - ✓ Content-based approaches are a generalization of topic-based approaches allowing the expression of subscriptions over a range of fields in an event notification.
- **Type-based:**
  - ✓ This is intrinsically linked with object-based approaches where objects have a specified type.
  - ✓ In type-based approaches, subscriptions are defined in terms of types of events and matching is defined in terms of types or subtypes of the given filter.

Implementation Issues

The publish subscribe model also demands of security, scalability, failure handling, concurrency and quality of service. This makes this model complex.

Centralized versus distributed implementations

- ✓ The centralized implementation of this model is forming a single node with a server on that node acting as an event broker.
- ✓ Publishers then publish events to this broker, and subscribers send subscriptions to the broker and receive notifications in return.
- ✓ Interaction with the broker is then through a series of point-to-point messages; this can be implemented using message passing or remote invocation.
- ✓ This lacks resilience and scalability, since the centralized broker represents a single point for potential system failure and a performance bottleneck.

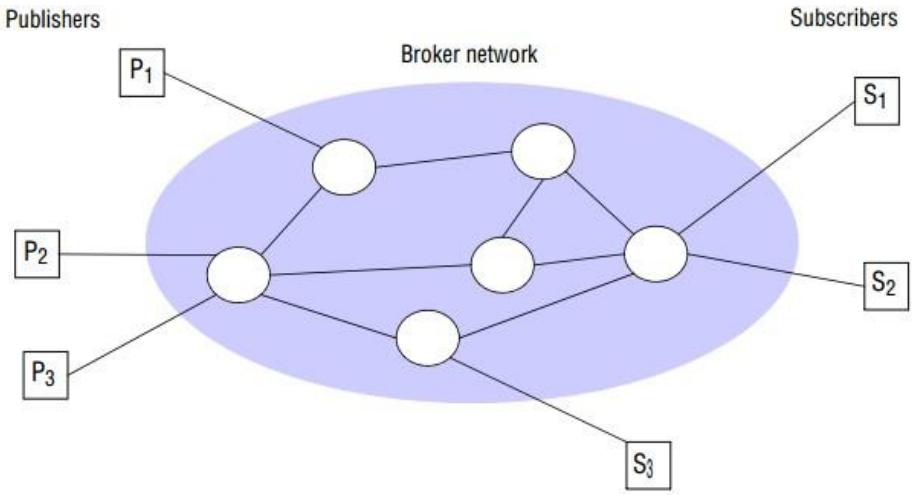


Fig 2.17: Network of Brokers

- ✓ Alternatively, the centralized broker is replaced by a network of brokers that cooperate to offer the desired functionality.
- ✓ A fully peer-to-peer implementation of a publish-subscribe system is also possible.
- ✓ In this approach, there is no distinction between publishers, subscribers and brokers; all nodes act as brokers, cooperatively implementing the required event routing functionality.

Overall systems architecture

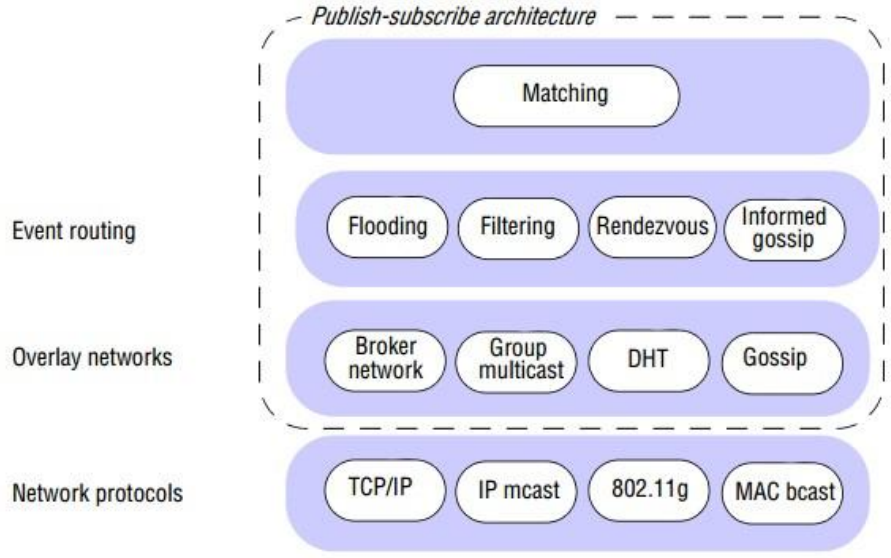


Fig 2.17: Overall system architecture

- ✓ In the bottom layer, publish-subscribe systems make use of a range of interprocess communication services.
- ✓ The event routing layer is supported by a network overlay infrastructure. Event routing performs the task of ensuring that event notifications are routed as efficiently as possible to appropriate subscribers
- ✓ The overlay infrastructure supports this by setting up appropriate networks of brokers or peer-to-peer structures.
- ✓ For content-based approaches, this problem is referred to as content-based routing (CBR), with the goal being to exploit content information to efficiently route events to their required destination.
- ✓ The top layer implements matching ensure that events match a given subscription.

The general principles behind content-based routing are:

- **Flooding:**
  - ✓ Sending an event notification to all nodes in the network and then carrying out the appropriate matching at the subscriber end is flooding.
  - ✓ Alternatively, flooding can also be used to send subscriptions back to all possible publishers, with the matching carried out at the publishing end and matched events sent directly to the relevant subscribers using point-to-point communication.
  - ✓ Flooding can be implemented using an underlying broadcast or multicast facility or arranging brokers in an acyclic graph in which each forwards incoming event notifications to all its neighbors.
- **Filtering:**
  - ✓ In filtering-based routing, filtering is done in the network of brokers.
  - ✓ Brokers forward notifications through the network only where there is a path to a valid subscriber.
  - ✓ This is achieved by propagating subscription information through the network towards potential publishers and then storing associated state at each broker.
  - ✓ Each node maintains a neighbors list containing a list of all connected neighbors in the network of brokers, a subscription list containing a list of all directly connected subscribers serviced by this node, and a routing table

- ✓ When a broker receives a publish request from a given node, it must pass this notification to all connected nodes where there is a corresponding matching subscription and also decide where to propagate this event through the network of brokers.
- ✓ This approach can cause heavy traffic due to propagation of subscription.

➢ **Advertisements:**

- ✓ Here the advertisements are propagated towards subscribers in a symmetrical way.

➢ **Rendezvous:**

- ✓ Here, the set of all possible events are viewed as an event space and responsibility is partitioned for this event space between the set of brokers in the network.
- ✓ This approach defines rendezvous nodes (broker nodes), that are responsible for a given subset of the event space.
- ✓ The rendezvous-based routing algorithm defines two functions.
  - SN(s) takes a given subscription(s), and returns one or more rendezvous nodes that take responsibility for that subscription. Each rendezvous node maintains a subscription list and forwards all matching events to the set of subscribing nodes.
  - When an event e is published, the function EN(e) also returns one or more rendezvous nodes, which are responsible for matching e against subscriptions in the system.
- ✓ Both SN(s) and EN(e) return more than one node if reliability is a concern.
- ✓ This approach works if the intersection of EN(e) and SN(s) is non-empty for a given e that matches s .

Rendezvous-based routing algorithm

```
upon receive publish(event e) from node x at node i
rvlist := EN(e);
if i in rvlistthen begin
matchlist<- match(e, subscriptions);
send notify(e) to matchlist;
end
send publish(e) to rvlist - i;
```

upon receive subscribe(subscription s) from node x at node i

rvlist := SN(s);

if i in rvliststhen

add s to subscriptions;

else

send subscribe(s) to rvlist - i;

2.12 MESSAGE QUEUES

Message Queuing technology enables applications running at different times to communicate across heterogeneous networks and systems that may be temporarily offline. Applications send messages to queues and read messages from queues. Message Queuing provides guaranteed message delivery, efficient routing, security, and priority-based messaging.

It can be used to implement solutions to both asynchronous and synchronous scenarios requiring high performance.

Programming Model

Communication in distributed systems is through queues. The producer processes can send messages to a specific queue and consumer processes can receive messages from this queue. There are three receiving styles

- blocking receive: block until an appropriate message is available
- non-blocking receive: check the status of the queue and return a message as available, or not available.
- Notify operation: issue an event notification when a message is available in the associated queue.

The queue can be build based on FIFO or priority.

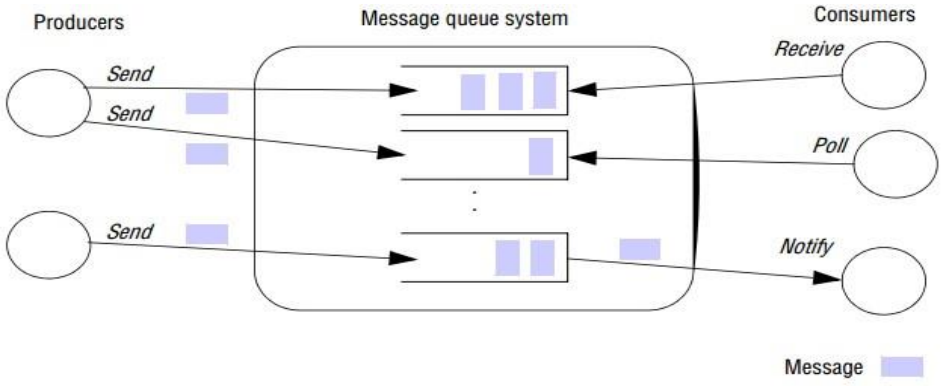


Fig 2.17: Message Queue

Properties of message passing system:

- Reliability
- Persistence
- Validity
- Integrity
- Guaranteed delivery
- Atomicity
- Transformation messages between formats to deal with heterogeneity in underlying data representations.
- Security

Implementation Issues

The important implementation decision for message queuing systems is the choice between centralized and distributed implementations.

WebSphere MQ

- ✓ This is a IBM developed middleware based on the concept of message queues.
- ✓ Queues in WebSphere MQ are managed by **queue managers** which host and manage queues and allow applications to access queues through the **Message Queue Interface**(MQI).
- ✓ This allows applications to carry out operations such as connecting to or disconnecting from a queue (MQCONN and MQDISC) or sending/receiving messages to/from a queue (MQPUT and MQGET).
- ✓ Multiple queue managers can reside on a single physical server. Client applications can also reside on the same physical server.
- ✓ The communication with the queue manager through is known as a **client channel.**
- ✓ MQI commands are issued on the proxy and then sent transparently to the queue manager for execution using RPC.
- ✓ The message channel is a unidirectional connection between two queue managers that is used to forward messages asynchronously from one queue to another.
- ✓ A message channel is managed by a message channel agent (MCA) at each end.
- ✓ Routing tables are also included in each queue manager, and together with channels this allows arbitrary topologies to be created.

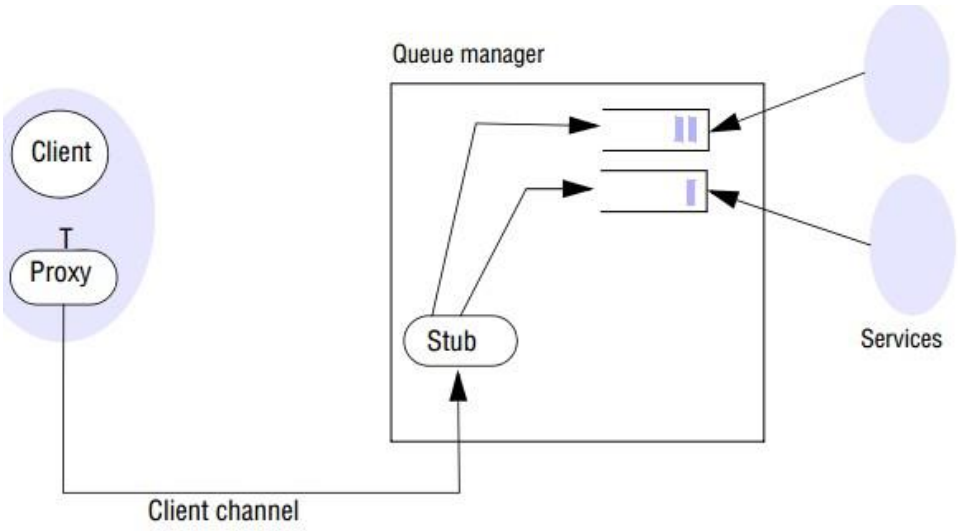


Fig 2.17: Network Topology in Websphere

The hub-and-spoke approach

- ✓ In the hub-and-spoke topology, one queue manager is designated as the hub.
- ✓ Client applications do not connect directly to this hub but rather connect through queue managers called **spokes.**
- ✓ Spokes relay messages to the message queue of the hub for processing by the various services.
- ✓ Spokes are placed strategically around the network to support different clients.
- ✓ The hub is placed on a node with sufficient resources to deal with the volume of traffic.
- ✓ This topology is heavily used with WebSphere MQ for large-scale deployments.
- ✓ The drawback of this architecture is that the hub can be a potential bottleneck and a single point of failure.

2.12.1 Java Messaging Services (JMS)

- ✓ The Java Messaging Service (JMS) is a standardized way for distributed Java programs to communicate indirectly.
- ✓ This unifies the publish-subscribe and message queue paradigms.
- ✓ A JMS client is a Java program or component that produces or consumes messages.

Programming with JMS

- ✓ Fir ough a cor
- ✓ Thi
- ✓ The Queue Co
  - ➢
  - ➢
- ✓ Co
- ✓ A n and cor

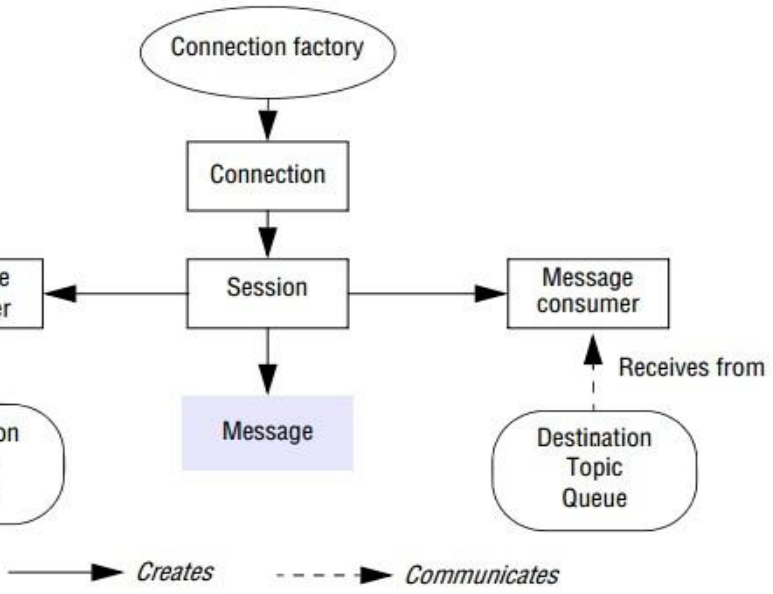


Fig 2.18: JMS Programming Model

Parts of a JMS Message

There are three parts: a header, a set of properties and the body of the message.

- **Header**
  - ✓ The header contains all the information needed to identify and route the message: destination, priority, expiration date, a message ID and a timestamp.

- ✓ These fields are either created by the underlying system, or constructor methods.
- ✓ The properties can be used to express additional context associated with the message, including a location field.

➢ **Body**

- ✓ The body is opaque and untouched by the system.
- ✓ The body can be any one of a text message, a byte stream, a serialized Java object, a stream of primitive Java values or a more structured set of name/value pairs.
- ✓ A message producer is an object used to publish messages under a particular topic or to send messages to a queue.
- ✓ A message consumer is an object used to subscribe to messages concerned with a given topic or to receive messages from a queue.
- ✓ The consumer is complicated:
  - Consumers apply message filters.
  - Consumer program either can block using a receive operation or it can establish a message listener object that identifies a message.

2.13 SHARED MEMORY APPROACHES

They are indirect communication paradigms that offer an abstraction of shared memory.

Distributed shared memory (DSM)

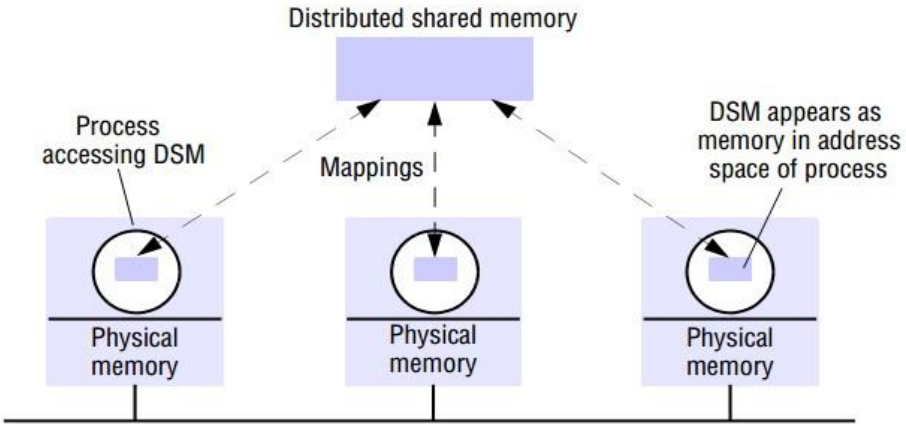


Fig 2.18: Distributed Shared Memory

- ✓ This is an abstraction used for sharing data between computers that do not share physical memory.
- ✓ Processes access DSM by reads and updates to what appears to be ordinary memory within their address space.
- ✓ The programmer need not worry about the message passing.
- ✓ DSM runtime support has to send updates in messages between computers.
- ✓ DSM systems must also manage replicated data.
- ✓ Each computer has a local copy of recently accessed data items stored in DSM, for speed of access.

Message passing versus DSM

The message passing and DSM can be compared based on services they offer and in terms of their efficiency.

Message Passing	**Distributed Shared Memory**
Services Offered: Variables have to be marshalled from one process, transmitted and unmarshalled into other variables at the receiving process.	The processes share variables directly, so no marshalling and unmarshalling. Shared variables can be named, stored and accessed in DSM.
Processes can communicate with other processes. They can be protected from one another by having private address spaces.	Here, a process does not have private address space. So one process can alter the execution of other.
This technique can be used in heterogeneous computers.	This cannot be used to heterogeneous computers.
Synchronization between processes is through message passing primitives.	Synchronization is through locks and semaphores.
Processes communicating via message passing must execute at the same time.	Processes communicating through DSM may execute with non-overlapping lifetimes.
Efficiency: All remote data accesses are explicit and therefore the programmer is always aware of whether a particular operation is in-process or involves the expense of communication.	Any particular read or update may or may not involve communication by the underlying runtime support.

2.13.1 Tuple space communication

Processes communicate indirectly by placing tuples in a tuple space, from which other processes can read or remove them.

The programming model

- ✓ Processes communicate through a shared collection of tuples.
- ✓ Any combination of types of tuples may exist in the same tuple space.
- ✓ Processes share data by accessing the same tuple space.
- ✓ The tuples are stored in tuple space using the write operation and read or extract them from tuple space using the reador take operation.
- ✓ No direct access to tuples in tuple space is allowed and processes have to replace tuples in the tuple space instead of modifying them. Thus, tuples are immutable.

Properties of tuple space

- ➢ **Space uncoupling:** A tuple placed in tuple space may originate from any number of sender processes and may be delivered to any one of a number of potential recipients.
- ➢ **Time uncoupling:** A tuple placed in tuple space will remain in that tuple space until removed and hence the sender and receiver do not need to overlap in time.

Implementation Issues

Centralized solution where the tuple space resource is managed by a single server is a common implementation.

Replication:

- ✓ A tuple space behaves like a state machine, maintaining state and changing this state in response to events received from other replicas or from the environment.
- ✓ To ensure consistency the replicas
  - (ii) Must start in the same state
  - (iii) must execute events in the same order
  - (iv) must react deterministically to each event.
- ✓ Alternatively, a multicast algorithm can be used.
- ✓ Here, updates are carried out in the context of the agreed set of replicas and tuples are also partitioned into distinct tuple sets based on their associated logical names.
- ✓ The system consists of a set of workers carrying out computations on the tuple space, and a set of tuple space replicas.

- ✓ A given physical node can contain any number of workers, replicas or indeed both; a given worker therefore may or may not have a local replica.
- ✓ Nodes are connected by a communications network that may lose, duplicate or delay messages and can deliver messages out of order.
- ✓ A write operation is implemented by sending a multicast message over the unreliable communications channel to all members of the view.
- ✓ The write request is repeated until all acknowledgements are received.
- ✓ The read operation consists of sending a multicast message to all replicas.

Write

1. The requesting site multicasts the write request to all members of the view;
2. On receiving this request, members insert the tuple into their replica and acknowledge this action;
3. Step 1 is repeated until all acknowledgements are received.

Read

1. The requesting site multicasts the read request to all members of the view;
2. On receiving this request, a member returns a matching tuple to the requestor;
3. The requestor returns the first matching tuple received as the result of the operation (ignoring others);
4. Step 1 is repeated until at least one response is received.

Phase 1: Selecting the tuple to be removed

1. The requesting site multicasts the take request to all members of the view;
2. On receiving this request, each replica acquires a lock on the associated tuple set and, if the lock cannot be acquired, the take request is rejected;
3. All accepting members reply with the set of all matching tuples;
4. Step 1 is repeated until all sites have accepted the request and responded with their set of tuples and the intersection is non-null;
5. A particular tuple is selected as the result of the operation .
6. If only a minority accept the request, this minority are asked to release their locks and phase 1 repeats.

Phase 2: Removing the selected tuple

1. The requesting site multicasts a remove request to all members of the view citing the tuple to be removed;

2. On receiving this request, members remove the tuple from their replica, send an acknowledgement and release the lock;

3. Step 1 is repeated until all acknowledgements are received

2.14 DISTRIBUTED OBJECTS

Distributed object communication realizes communication between distributed objects in the distributed computing environment. The main role is to interconnect objects residing in non-local memory space and allow them to perform remote calls and exchange data. Distributed object middleware offers a programming abstraction based on object oriented principles.

The following table gives a brief description about relation between objects and distributed objects:

Objects	**Distributed Objects**	**Descriptions**
Object references	Remote object references	Unique reference for each distributed object.
Interfaces	Remote interfaces	Provides methods to be invoked o remote object using DLL.
Actions	Distributed actions	Invocation of methods
Exceptions	Distributed Exception	Includes exceptions generated in distributed environment
Garbage Collection	Distributed Garbage Collection	Implements distributed garbage collection algorithm

Differences between objects and distributed objects

Objects	**Distributed Objects**
Class is a fundamental concept in object-oriented languages.	Classes are not prominent here.
In the object oriented world, class can be the description of the behavior associated with a group of objects or, the place to go to instantiate an object with a given behavior or even the group of objects that adhere to that behavior.	Here the term „class“ is avoided, more specific terms such as „factory“ and „template“ are readily used.
Object oriented languages offer implementation inheritances	This offers interface inheritances.

Additional Complexities

- **Inter-object communication:**
  - ✓ This is normally provided by remote method invocation.
- **Lifecycle management**:
  - ✓ Lifecycle management is concerned with the creation, migration and deletion of objects, with each step having to deal with the distributed nature of the underlying environment.
- **Activation and deactivation:**
  - ✓ In distributed systems, the numbers of objects may be very large, and hence it would be waste of resources to have all objects available (active) at any time.
  - ✓ Activation is the process of making an object active in the distributed environment by providing the necessary resources for it to process incoming invocations.
  - ✓ Deactivation is rendering an object temporarily unable to process invocations.
- **Persistence:**
  - ✓ Objects have state, and it maintains this state across possible cycles of activation and deactivation and indeed system failures.
  - ✓ Distributed object middleware must therefore offer persistency management for stateful objects.
- **Additional services:**
  - ✓ This must provide support for the range of distributed system services considered in this book, including naming, security and transaction services.

2.15 FROM OBJECTS TO COMPONENTS

Component based computing is out powering distributed objects.

Problems in object-oriented middleware

The following four problems in object oriented middleware led to the development of component based technology:

- **Implicit dependencies:**
  - ✓ A distributed object communicates with outside world through interfaces.
  - ✓ The interfaces provide a complete contract for the deploying and use of this object.

- ✓ The problem in distributed objects is that the internal behavior of an object is hidden.
- ✓ Implicit dependencies in the distributed configuration are not safe.
- ✓ The third-party developers to implement one particular element in a distributed configuration.

  Requirement: Dependencies of the object with otherobjects in the distributed configuration must be specified.

➢ **Interaction with the middleware:**

- ✓ The distributed objects must interact even with low-levelmiddleware architecture.

  Requirement: While programming distributed applications, a clean separation must be made between code related to middleware framework and code associated with the application.

➢ **Lack of separation of distribution concerns:**

- ✓ Distributed programmers must also focus on issues related to security, transactions, coordination and replication.
- ✓ To provide the non functional requirements :
  - Programmers must have knowledge of the full details of all the associated distributed system services.
  - The implementation of an object will contain calls to distributed system services and to middleware interfaces. This increases the programming complexity.

  Requirement: These complexities should be hidden from the programmer.

➢ **No support for deployment:**

- ✓ Objects must be deployed manually on individual machines and activated when required.
- ✓ This is a tiresome and error-prone process.

  Requirement: Middleware platforms should support deployment.

2.15.1 Fundamentals of Components

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only.

The term contract refers to:

- a set of provided interfaces
- a set of required interfaces

Every required interface must be bound to a provided interface of another component. This is called **software architecture.** This consists of components, interfaces and connections between interfaces.

Programming in component-based systems is concerned with the development of components and their composition. So third party development of software components is possible in **component based technology (CBT).**

Advantages of CBT

- ✓ Independent extensions
- ✓ Component Market
- ✓ Component models lessen unanticipated interactions between components
- ✓ Reduced time to market
- ✓ Reduced Costs

Disadvantages of CBT:

- ✓ Time to develop software components takes a big effort.
- ✓ Components can be pricey.
- ✓ Requirements in component technologies lacking
- ✓ Conflict between usability and reusability of components.
- ✓ Maintenance cost for components increased

Components and distributed systems Containers:

- ✓ The containers are absolutely central to component-based middleware.
- ✓ Containers support a common pattern often encountered in distributed applications, which consists of:
  - o a front-end client
  - o a container holding one or more components that implement the application or business logic
  - o system services that manage the associated data in persistent storage.

- ✓ The containers provide a managed server-side hosting environment for components.
- ✓ The components deal with application concerns and the container deals with distributed systems and middleware issues.
- ✓ A container includes a number of components that require the same configuration in terms of distributed system support.

Support for Deployment

- ✓ Component-based middleware supports the deployment of component configurations.
- ✓ They include releases of software architectures (components and their interconnections) together with deployment descriptors.
- ✓ These descriptors describe how the configurations should be deployed in a distributed environment.
- ✓ Deployment descriptors are typically written in XML and include sufficient information to ensure that:
  - o components are correctly connected using appropriate protocols and associated middleware support
  - o the underlying middleware and platform are configured to provide the right level of support to the component configuration
  - o the associated distributed system services are set up to provide the right level of security, transaction support and so on.

2.16 ENTERPRISE JAVA BEANS

- ✓ Enterprise JavaBeans (EJB) is a specification of a server-side, managed component architecture
- ✓ It supports the development of the classic style of application, where potentially large numbers of clients interact with a number of services realized through components or configuration of components.
- ✓ The components, which are known as beans in EJB, are intended to capture the application (or business) logic
- ✓ The container in EJB issues calls to the associated services to provide the required properties.
- ✓ Beans act as:
  - ➢ Container Managed: The transaction manager or security services is completely hidden from the developer of the associated beans.

- Bean Managed: The bean developer takes control over these operations not the transaction manager.

✓ The EJB specification has the following roles:

- bean provider: develops the application logic of the component(s)
- application assembler: assembles beans into application configurations
- deployer: takes a given application assembly and ensures it can be correctly deployed in a given operational environment;
- service provider: does transaction management
- persistence provider: maps persistent data to databases
- container provider: build on the above two roles and is responsible for configuring containers with the required level of distributed systems support in terms of non-functional properties
- system administrator: responsible for monitoring a deployment at runtime and making any adjustments to ensure its correct operation.

The EJB component model

✓ A bean in EJB is a component offering one or more business interfaces to potential clients of that component.

✓ A bean is represented by the set of remote and local business interfaces together with an associated bean class that implements the interfaces.

✓ Two main styles of bean are supported in the EJB specification:

- **Session beans:** A session bean is a component implementing a particular task within the application logic of a service.
- **Message-driven beans:** Clients interact with session beans using local or remote invocation.

POJOs and annotations

The task of programming in EJB has been simplified significantly through the use of Enterprise JavaBean POJOs(plain old Java objects). It consists of the application logic written simply in Java with no other code relating to it being a bean. Annotations are then used to ensure the correct behaviour in the EJB context. **Enterprise JavaBean containers in EJB**

✓ Beans are deployed to containers, and the containers provide implicit distributed system management.

✓ The container provides the policies in areas including transaction management, security, persistence and lifecycle management.

- ✓ The developer needs to focus only on application logic.
- ✓ EJB manages transactions.
- ✓ Transactions are sequences of operations, and that the sequences must be identified by the transaction manager.
- ✓ The first thing to declare is whether transactions associated with an enterprise bean should be bean-managed or container-managed. @TransactionManagement (BEAN)@TransactionManagement (CONTAINER)
- ✓ The bean managed transaction uses the following methods:
  - ➢ javax.transaction.UserTransaction– the User.Transaction.begin
  - ➢ UserTransaction.commitmethods – within the code of the bean.

Transaction Attributes in EJB

Attribute	**Policy**
REQUIRED	If the client has an associated transaction running, execute within this transaction; otherwise, start a new transaction.
REQUIRES_NEW	Always start a new transaction for this invocation.
SUPPORTS	If the client has an associated transaction, execute the method within the context of this transaction; if not, the call proceeds without any transaction support.
NOT_SUPPORTED	If the client calls the method from within a transaction, then this transaction is suspended before calling the method and resumed afterwards – that is, the invoked method is excluded from the transaction.
MANDATORY	The associated method must be called from within a client transaction; if not, an exception is thrown.
NEVER	The associated methods must not be called from within a client transaction; if this is attempted, an exception is thrown

Dependency injection:

- ✓ In dependency injection a third party (container), is responsible for managing and resolving the relationships between a component and its dependencies.
- ✓ A component refers to a dependency using an annotation and the container is responsible for resolving this annotation and ensuring that, at runtime, the associated attribute refers to the right object.
- ✓ This is typically implemented by the container usingreflection.

Enterprise JavaBean Interception

The Enterprise JavaBeans allows interception of two types of operation to alter their default behavior:

- method calls associated with a business interface
- life cycle events.

Invocation Contexts

Method	**Description**
public Object getTarget()	Returns the bean instance associated with the incoming invocation or event
public Method getMethod()	Returns the method being invoked
public Object[] getParameters()	Returns the set of parameters associated with the intercepted business method
public void setParameters (Object[] params)	Allows the parameter set to be altered by theinterceptor, assuming type correctness ismaintained
public Object proceed() throws Exception	Execution proceeds to next interceptor in the chain (if any) or the method that has been intercepted

REVIEW QUESTIONS

PART-A

1. What is IPC?

Interprocess communication (IPC) is a set of programming interfaces that allows a programmer to coordinate activities among different program processes that can run concurrently in an operating system.

2. List the models in distributed system.

- Physical models: It is the most explicit way in which to describe a system in terms of hardware composition.
- Architectural models: They describe a system in terms of the computational and communication tasks performed by its computational elements.
- Fundamental models: They examine individual aspects of a distributed system. They are again classified based on some parameters as follows:
  - ✓ Interaction models: This deals with the structure and sequencing of the communication between the elements of the system
  - ✓ Failure models: This deals with the ways in which a system may fail to operate correctly
  - ✓ Security models: This deals with the security measures implemented in the system against attempts to interfere with its correct operation or to steal its data.

3. Define ULS.

The ultra large scale (ULS) distributed systems is 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 or tasks.

4. What are the characteristics of ULS?

- very large size
- global geographical distribution
- operational and managerial independence of their member systems.

5. What are the evaluation elements of architectural model?

Architectural elements, architectural patterns and middleware platforms.

6. What are the architectural entities?

- Communicating entities (objects, components and web services);
- Communication paradigms (inter process communication, remote invocation and indirect communication);
- Roles responsibilities and placement

7. What are the communicating entities in architectural model?

- ↑ Objects: They are used to implement object oriented approaches in distributed systems. Objects are accessed through interfaces.
- ↑ Components: Components resemble objects and they offer problem-oriented abstractions for building distributed systems. They are also accessed through interfaces. The key difference between component and object is that a components holds all the assumptions specified to other components and their interfaces in the system. Components and objects are used to develop tightly coupled applications.
- ↑ Web services: Web services are closelyrelated to objects and components. Web services areintegrated into the World Wide Web. They are partially defined by the web-based technologies theyadopt. Web services are generally viewed as complete servicesthat can be combined to achieve value-added services across organizational boundaries.

8. List the communication mechanisms in DS.

IPC, Remote invocation and indirect communication.

9. Give the features of IPC.

This has the following two key features:

- ✓ Space uncoupling: Senders do not need to know who they are sending to
- ✓ Time uncoupling: Senders and receivers do not need to exist at the same time

10. What is 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. A group identifier uniquely identifies the group. The recipients join the group and receive the messages. Senders send messages to the group based on the group identifier and hence do not need to know the recipients of the message.

11. What are Publish-subscribe systems?

This is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers. Instead, published messages are characterized into classes, without knowledge of what, subscribers there may be. Similarly, subscribers express interest in one or more classes, and only receive messages that are of interest, without knowledge of what, if any, publishers there are. They offer one-to-many style of communication.

12. What are Message queues?

They 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. Queues therefore offer an indirection between the producer and consumer processes.

13. What are Tuple spaces?

The 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. This style of programming is known as generative communication.

14. Write about Distributed shared memory.

In computer architecture, distributed shared memory (DSM) is a form of memory architecture where the (physically separate) memories can be addressed as one (logically shared) address space. Here, the term shared does not mean that there is a single centralized memory but shared essentially means that the address space is shared (same physical address on two processors refers to the same location in memory

15. What are the architectures based on roles and responsibilities?

Client server and peer to peer.

16. What are the strategies for placement of objects?

- Mapping of services to multiple servers
- Caching
- Mobile code
- Mobile agents

17. What is mobile code?

Applets are a well-known and widely used example of mobile code. The user running a browser selects a link to an applet whose code is stored on a web server; the code is downloaded to the browser and runs. They provide interactive response.

18. What are mobile agents?

A mobile agent is a running program (including both code and data) that travels from one computer to another in a network carrying out a task on someone‟s behalf, such as collecting information, and eventually returning with the results. A mobile agent is a complete program, code + data, that can work (relatively) independently.

19. What is layering?

In a layered approach, a complex system is partitioned into a number of layers, with a given layer making use of the services offered by the layer below.

20. Define tiering.

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.

21. Write about two tier architecture.

The two tiered architecture refers to client/server architectures in which the user interface (presentation layer) runs on the client and the database (data layer) is stored on the server. The actual application logic can run on either the client or the server.

22. Write about three tier architecture.

Presentation tier: This is the topmost level of the application. is concerned with handling user interaction and updating the view of the application as presented to the user;

Application tier (business logic, logic tier, or middle tier): The logical tier is pulled out from the presentation tier and, as its own layer, it controls an application‟s functionality by performing detailed processing.

Data tier: The data tier includes the data persistence mechanisms (database servers, file shares, etc.) and the data access layer that encapsulates the persistence mechanisms and exposes the data.

23. Define thin client.

In a thin-client model, all of the application processing and data management is carried out on the server. The client is simply responsible for running the responsible for running the presentation software. This is used when legacy systems are migrated to client server architectures.

24. What is VNC?

The virtual network computing (VNC) has emerged to overcome the disadvantages of thin clients. VNC is a type of remote-control software that makes it possible to control another computer over a network connection.

25. What is Proxy?

- This facilitates location transparency in remote procedure calls or remote method invocation.
- A proxy is created in the local address space to represent the remote object with same interface as the remote object.
- The programmer makes calls on this proxy object and he need not be aware of the distributed nature of the interaction.

26. Define Brokerage.

- It is used to bring interoperability in potentially complex distributed infrastructures.
- The service broker is meant to be a registry of services, and stores information about what services are available and who may use them.

27. What is reflection?

- The Reflection architectural pattern provides a mechanism for changing structure and behaviour of software systems dynamically.
- It supports the modification of fundamental aspects, such as type structures and function call mechanisms.

28. What are the levels in reflection?

Meta Level: This provides information about selected system properties and makes the software self-aware.

Base level: This includes the application logic. Its implementation builds on the meta level. Changes to information kept in the meta level affect subsequent base-level behavior.

29. Define middleware.

Middleware is a general term for software that serves to "glue together" separate, often complex and already existing, programs.

30. What are Distributed Objects?

The term distributed objects usually refers to software modules that are designed to work together, but reside either in multiple computers connected via a network or in different processes inside the same computer. One object sends a message to another object in a remote machine or process to perform some task. The results are sent back to the calling object.

31. What are Distributed Components?

A component is a reusable program building block that can be combined with other components in the same or other computers in a distributed network to form an application. Examples: a single button in a graphical user interface, a small interest calculator, an interface to a database manager. Components can be deployed on different servers in a network and communicate with each other for needed services. A component runs within a context called a container . Examples: pages on a Web site, Web browsers, and word processors.

32. Write about Publish subscriber model.

Publish–subscribe is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called

subscribers. Instead, published messages are characterized into classes, without knowledge of what, if any, subscribers there may be. Similarly, subscribers express interest in one or more classes, and only receive messages that are of interest, without knowledge of what publishers are there.

33. Write about Message Queues.

Message queues provide an asynchronous communications protocol, meaning that the sender and receiver of the message do not need to interact with the message queue at the same time. Messages placed onto the queue are stored until the recipient retrieves them. Message queues have implicit or explicit limits on the size of data that may be transmitted in a single message and the number of messages that may remain outstanding on the queue.

34. Define Web services.

A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

35. What is Peer to peer computing?

Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or work load between peers. Peers are equally privileged, equipotent participants in the application. They are said to form a peer-to-peer network of nodes.

36. List the advantages of middleware

- ✓ Real time information access among systems.
- ✓ Streamlines business processes and helps raise organizational efficiency.
- ✓ Maintains information integrity across multiple systems.
- ✓ It covers a wide range of software systems, including distributed Objects and components, message-oriented communication, and mobile application support.
- ✓ Middleware is anything that helps developers create networked applications.

37. List the disadvantages of middleware

- ✓ Prohibitively high development costs.
- ✓ There are only few people with experience in the market place to develop and use a middleware.

- ✓ There are only few satisfying standards.
- ✓ The tools are not good enough.
- ✓ Too many platforms to be covered.
- ✓ Middleware often threatens the real-time performance of a system.
- ✓ Middleware products are not very mature.

38. Define fundamental models.

Fundamental Models are concerned with a formal description of the properties that are common in all of the architectural models.

39. What are the types of fundamental models?

Interaction Model – deals with performance and the difficulty of setting of time limits in a distributed system.

Failure Model – specification of the faults that can be exhibited by processes

Secure Model – discusses possible threats to processes and communication channels.

40. Give the purpose of the fundamental model.

- ✓ Make explicit all the relevant assumptions about the systems we are modelling.
- ✓ Make generalizations (general purpose algorithms) concerning what is possible or impossible with given assumptions.

41. What are the communication performance metrics?

a. Latency: A delay between the start of a message‟s transmission from one process to the beginning of reception by another.

b. Bandwidth: The total amount of information that can be transmitted over in a given time. The communication channels using the same network, have to share the available bandwidth.

c. Jitter: The variation in the time taken to deliver a series of messages. It is very relevant to multimedia data.

42. What is synchronous DS?

- ♦ This is practically hard to achieve in real life.
- ♦ In synchronous DS the time taken to execute a 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 known bound.

43. What is Asynchronous DS?

- There are no bounds on: process execution speeds, message transmission delays and clock drift rates.

44. What are the common types of failures?

- Omission Failure
- Arbitrary Failure
- Timing Failure

45. What is Omission failure?

Omission failures occur due to communication link failures. They are detected through timeouts.

46. **Write about Process omission failures**.

- ✓ Omission failures that occur due to process crash (i.e.) the execution of the process could not continue.
- ✓ This is detected using timeouts.
- ✓ A fixed period of time is fixed for all the methods to complete its execution. If the method takes time longer than the allowed time, a time out has occurred.
- ✓ In an asynchronous system a timeout can indicate only that a process is not responding.
- ✓ A process crash is called fail-stopif other processes can detect certainly that the process has crashed.
- ✓ Fail-stop behavior can be produced in a synchronous system if the processes use timeouts to detect when other processes fail to respond and messages are guaranteed to be delivered.

47. Write about Communication omission failures.

- ✓ This occurs when the messages are dropped between sender and the receiver.
- ✓ The message can be lost in sender buffer, receiver buffer or even in the communication channel.

48. What is send omission failure?

The loss of messages between the sending process and the outgoing message buffer is known as send omission failures.

49. What is receive-omission failure?

The loss of messages between the incoming message buffer and the receiving process as receive-omission failures.

50. What is channel omission failure?

The loss of messages in a channel is channel omission failure.

51. What are Arbitrary failures (Byzantine failure)?

This includes all possible errors that could cause failure.

52. Classify the arbitrary failures.

Class	Affects	Comments
Fail stop	Process	Process halts and remains halted. Other processes may detect that this process has halted.
Crash	Process	Process halts and remains halted. Other processes may not detect that this process has halted.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end‟s incoming messagebuffer.
Send-omission	Process	A process completes a send operation but the message is not put in its outgoing message buffer.
Receiveomission	Process	A message is put in a process‟s incoming messagebuffer, but that process does not receive it.
Arbitrary	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times or commit omissions; a process may stop or take an incorrect step.

53. Define timing failure.

Timing failures refer to a situation where the environment in which a system operates does not behave as expected regarding the timing assumptions, that is, the timing constraints are not met.

54. Define reliable communication.

The term reliable communication is defined in terms of validity and integrity.

- 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.

55. What is security model?

The security of a DS can be achieved by securing the processes and the channels used in their interactions and by protecting the objects that they encapsulate against unauthorized access.

56. What is DOS attack?

Denial of service (DoS) attack is an incident in which a user or organization is deprived of the services of a resource they would normally expect to have. In a distributed denial-of-service, large numbers of compromised systems (sometimes called a botnet) attack a single target.

57. Define IPC.

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.

58. What are the characteristics of IPC?

Characteristics of IPC:

- Synchronous and asynchronous communication
- Message destinations
- Reliability
- Ordering

59. Define socket.

A socket is one endpoint of a two-way communication link between two programs unning on the network.

60. What are the UDP failure modes?

UDP datagrams suffer from following failures:

- Omission failure: Messages may be dropped occasionally
- Ordering: Messages can be delivered out of order.

61. List the issues in stream communication.

Matching of data items: Two communicating processes need to agree as to the contents of the data transmitted over a stream.

Blocking: The process that writes data to a stream may be blocked by the TCP flow-control mechanism if the socket at the other end is queuing as much data as the protocol allows.

Threads: When a server accepts a connection, it generally creates a new thread in which to communicate with the new client. In an environment in which threads are not provided, an alternative is to test whether input is available from a stream before attempting to read it;

62. Define external data representation.

External Data Representation is an agreed standard for the representation of data structures and primitive values.

63. Define marshalling and unmarshalling.

Marshalling is the process of taking a collection of data items and assembling them into a form suitable for transmission in a message. Unmarshalling is the process of disassembling a collection of data on arrival to produce an equivalent collection of data items at the destination.

64. Define reflection.

Reflection is inquiring about class properties, e.g., names, types of methods and variables, of objects

65. List the requirements of remote object reference.

The following are mandatory for remote object reference

- internet address/port number: process which created object
- time: creation time
- object number: local counter, incremented each time an object is created in the creating process
- interface: how to access the remote object (if object reference is passed from one client to another)

66. Define multi cast communication.

Multicast (one-to-many or many-to-many distribution) is group communication where information is addressed to a group of destination computers simultaneously.

67. List the characteristics of multicasting.

The following are the characteristics of multicasting:

- Fault tolerance based on replicated services:
- Finding the discovery servers in spontaneous networking
- Better performance through replicated data
- Propagation of event notifications

68. What is IP multicast?

IP multicast is built on top of the Internet protocol. 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 with1110 as it starting byte.

69. List some methods in MulticastSocket class.

- getInterface():Retrieve the address of the network interface used for multicast packets.
- getTTL():Get the default time-to-live for multicast packets sent out on the socket.
- joinGroup(InetAddress): Joins a multicast group.
- leaveGroup(InetAddress):Leave a multicast group.
- send(DatagramPacket, byte):Sends a datagram packet to the destination, with a TTL (time- to-live) other than the default for the socket.
- setInterface(InetAddress): Set the outgoing network interface for multicast packets on this socket, to other than the system default.
- setTTL(byte): Set the default time-to-live for multicast packets sent out on this socket.

70. Define network virtualization.

Network virtualization refers to the management and monitoring of an entire computer network as a single administrative entity from a single software-based administrator‟s console.

71. Define overlay network.

An overlay network is a virtual network of nodes and logical links that is built on top of an existing network with the purpose to implement a network service that is not available in the existing network.

72. List the Advantages of overlay network.

- ✓ They enable new network services to be defined without requiring changes to the underlying network.
- ✓ 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.

73. List the disadvantages of overlay network.

- ✓ They introduce an extra level of indirection.
- ✓ They add to the complexity of network services

74. Define message passing interface.

Message Passing Interface (MPI) is a standardized and portable message-passing system to function on a wide variety of parallel computers.

75. What are the types of message passing?

- ➢ Synchronous message passing: This is implemented by blocking send and receive calls.
- ➢ Asynchronous message passing: This is implemented by a non-blocking form of send.

76. What is remote invocation?

Remote invocations are method invocations for objects in different processes.

77. What are remote objects?

Remote objects are objects that can receive remote invocation. The core part of the distributed object model is:

- remote object reference;
- remote interface: specifies which methods can be invoked remotely;

78. Give the advantages of UDP over TCP.

UDP is preferred over TCP for the following reasons:

- ♦ Acknowledgements are redundant, since requests are followed by replies.
- ♦ Establishing a connection involves two extra pairs of messages in addition to the pair required for a request and a reply.
- ♦ Flow control is redundant for the majority of invocations, which pass only small arguments and results.

79. What is message id?

Each message have a unique message identifier for referencing it. A message identifier consists of two parts:

- request Id: Increasing sequence of integers assigned by the sending process. This makes the message unique to the sender.
- an identifier: for the sender process. This makes the message unique in the distributed system.

80. What is history of messages?

History is used to refer to a structure that contains a record of reply messages that have been transmitted by the server.

81. List the styles of exchange protocols.

There are three exchange protocols:

- request (R) protocol: a single request message is sent by the client to the server
- request-reply (RR) protocol: used in client-server exchanges because it is based on the request-reply protocol. Special acknowledgement messages are not required.
- request-reply-acknowledge reply (RRA) protocol: It is based on request-reply-acknowledge reply. The Acknowledge reply message contains the requested from the reply message being acknowledged. This will enable the server to discard entries from its history. The arrival of a requested in an acknowledgement message will be interpreted as acknowledging the receipt of all reply messages with lower request Ids.

82. Give the differences between persistent and non persistent connections

Persistent Connection	Non persistent Connection
On the same TCP connection the server, parses request, responds and waits for new requests.	The server parses request, responds, and closes TCP connection.
It takes fewer RTTs to fetch the objects.	It takes 2 RTTs to fetch each object.
It has less slow start.	Each object transfer suffers from slow start

83. Define RPC.

Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details.

84. List the design issues of RPC.

The following are the design issues in RPC:

- ❖ Programming with interfaces
- ❖ Interfaces in distributed systems
- ❖ Interface definition languages:
- ❖ RPC call semantics

85. Define stub.

A stub is a piece of code that is used to convert parameters during a remote procedure call (RPC). An RPC allows a client computer to remotely call procedures on a server computer.

86. Define RMI.

RMI is a set of protocols being developed by Sun's JavaSoft division that enables Java objects to communicate remotely with other Java objects.

87. List the differences between RMI and RPC

RMI	RPC
RMI uses an object oriented paradigm where the user needs to know the object and the method of the object he needs to invoke.	RPC is not object oriented and does not deal with objects. Rather, it calls specific subroutines that are already established
With RPC looks like a local call. RPC handles the complexities involved with passing the call from the local to the remote computer.	RMI handles the complexities of passing along the invocation from the local to the remote computer. But instead of passing a procedural call, RMI passes a reference to the object and the method that is being called.

88. Give the Limitations of RMI

- ✓ RMI is not language independent. It is limited only to Java.
- ✓ Interfaces to remote objects are defined using ordinary Java interfaces not with special IDLs.

89. Give the differences between Interfaces and Class

Interfaces	Class
Interfaces cannot be instantiated.	Classes can be instantiated.
They do not contain constructors.	They can have constructors.
They cannot have instance fields (i.e.) all the fields in an interface must be declared both static and final.	No constraints over the fields.
The interface can contain only abstract method.	They can contain method implementation.
Interfaces are implemented by a class not extended.	Classes are extended by other classes.

90. What is remote object reference?

A remote object reference is an identifier that can be used throughout a distributed system to refer to a particular unique remote object.

91. Give the differences between remote and local objects

Remote Objects	Local Objects
Creating a remote object requires creating a stub and a skeleton, which will cost more time cycles.	They consume less time cycles.
Marshalling and unmarshalling is required.	Marshalling and unmarshalling is not required.
Longer time is taken by the remote object to return to the calling program.	Comparatively shorter return time.

92. What is reference counting?

Distributed garbage collection is usually based on reference counting. The total number of references to the object is maintained by a *reference count* field.

93. Give the use of Proxy in RMI.

The class of the proxy contains a method corresponding to each method in the remote interface which marshalls arguments and unmarshalls results for that method. It hides the details of the remote object reference. Each method of the proxy marshals a reference to the target object using a request message (operationId, arguments). When the reply is received the proxy, unmarshals it and returns the results to the invoker.

94. Give the use of Dispatcher in RMI.

Server has a dispatcher & a skeleton for each class of remote object. The dispatcher receives the incoming message, and uses its method info to pass it to the right method in the skeleton.

95. Give the use of Skeleton in RMI.

Skeleton implements methods of the remote interface to unmarshall arguments and invoke the corresponding method in the servant. It then marshalls the result (or any exceptions thrown) into a reply message to the proxy.

96. What are factory methods?

Factory methods are static methods that return an instance of the native class.Factory method is used to create different object from factory often refereed as Item and it encapsulate the creation code.

97. **List the responsibilities of activator**.

Responsibilities of an activator:

- ✓ Registering passive objects that are available for activation. This is done by mapping the names of servers and passive objects.
- ✓ Starting named server processes and activating remote objects in them.
- ✓ Tracking the locations of the servers for remote objects that it has alreadyactivated.
- ✓ Java RMI provides the ability to make some remote objects activatable [java.sun.comIX].

98. Define persistent object.

An object that is guaranteed to live between activations of processes is called a persistent object.

99. What is Jini?

Jini provides a mechanism for locating services on the network that conform to a particular (Java) interface, or that have certain attributes associated with them. Once a service is located, the client can download an implementation of that interface, which it then uses to communicate with the service.

100. What is callback?

Callbackrefers to a server‟s action of notifying clients about an event.

101. Define group communication.

Group communication is a service where a message is sent to a group and then this message is delivered to all members of the group.

102. What is publish subscribe model?

Publish–subscribe is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers.

103. What is message queuing?

Message Queuing technology enables applications running at different times to communicate across heterogeneous networks and systems that may be temporarily offline.

104. Define session.

A session is a series of operations involving the creation, production and consumption of messages related to a logical task.

105. What are the parts of JMS message?

There are three parts: a header, a set of properties and the body of the message.

106. What is DSM?

This is an abstraction used for sharing data between computers that do not share physical memory. Processes access DSM by reads and updates to what appears to be ordinary memory within their address space.

107. Give the differences between message passing and DSM.

Message Passing	Distributed Shared Memory
Services Offered: Variables have to be marshalled from one process, transmitted and unmarshalled into other variables at the receiving process.	The processes share variables directly, so no marshalling and unmarshalling. Shared variables can be named, stored and accessed in DSM.
Processes can communicate with other processes. They can be protected from one another by having private address spaces.	Here, a process does not have private address space. So one process can alter the execution of other.
This technique can be used in heterogeneous computers.	This cannot be used to heterogeneous computers.
Synchronization between processes is through message passing primitives.	Synchronization is through locks and semaphores.
Processes communicating via message passing must execute at the same time.	Processes communicating through DSM may execute with non-overlapping lifetimes.

Efficiency: All remote data accesses are explicit and therefore the programmer is always aware of whether a particular operation is in-process or involves the expense of communication.	Any particular read or update may or may not involve communication by the underlying runtime support.

108. Give the properties of tuple space.

- Space uncoupling: A tuple placed in tuple space may originate from any number of sender processes and may be delivered to any one of a number of potential recipients.
- Time uncoupling: A tuple placed in tuple space will remain in that tuple space until removed and hence the sender and receiver do not need to overlap in time.

109. Give the differences between objects and distributed objects.

Objects	Distributed Objects
Class is a fundamental concept in object-oriented languages.	Classes are not prominent here.
In the object oriented world, class can be the description of the behavior associated with a group of objects or, the place to go to instantiate an object with a given behavior or even the group of objects that adhere to that behavior.	Here the term „class" is avoided, more specific terms such as „factory" and „template" are readily used.
Object oriented languages offer implementation inheritances	This offers interface inheritances.

110. Define software component.

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only.

111. Give the advantages and disadvantages of CBT.

Advantages of CBT

- ✓ Independent extensions

- ✓ Component Market
- ✓ Component models lessen unanticipated interactions between components
- ✓ Reduced time to market
- ✓ Reduced Costs

Disadvantages of CBT:

- ✓ Time to develop software components takes a big effort.
- ✓ Components can be pricey.
- ✓ Requirements in component technologies lacking
- ✓ Conflict between usability and reusability of components.
- ✓ Maintenance cost for components increased

112. What is EJB?

Enterprise JavaBeans (EJB) is a specification of a server-side, managed component architecture. It supports the development of the classic style of application, where potentially large numbers of clients interact with a number of services realized through components or configuration of components.

PART-B

1. Describe the physical system model.
2. Explain architectural models.
3. Write notes about architectural patterns.
4. Explain the middleware.
5. Describe the fundamental models.
6. Explain about interprocess communication.
7. Describe external data representation and marshaling.
8. Elucidate multicast communication.
9. Explain overlay networks.
10. Write notes on message passing interface.
11. Describe remote invocation.
12. Write in detail about Java RMI.

13. Explain group communications.
14. Describe publish subscriber systems.
15. Write about message queues.
16. Explain shared memory approach.
17. Describe distributed objects.
18. How to convert from objects to components?
19. Write about EJB.

3

PEER TO PEER SERVICES AND FILE SYSTEM

3.1 INTRODUCTION TO PEER TO PEER SYSTEMS

Peer-to-peer (P2P) is a decentralized communications model in which each party has the same capabilities and either party can initiate a communication session.

Unlike the client/server model, in which the client makes a service request and the server fulfills the request, the P2P network model allows each node to function as both client and server.

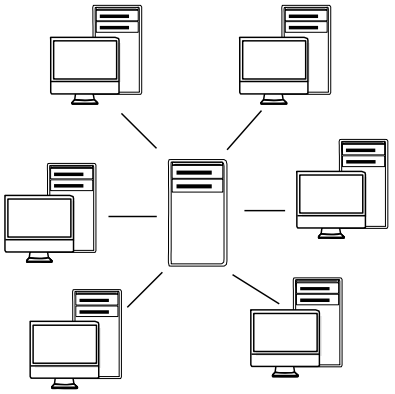
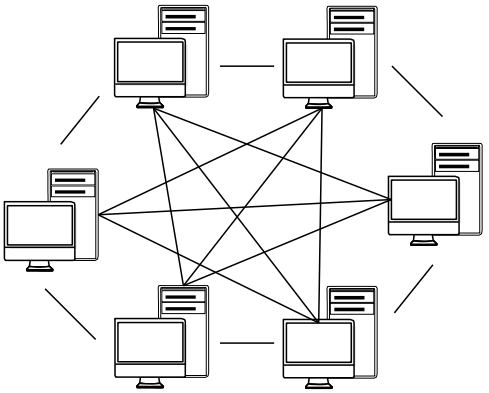
Server Based Communication	Peer to Peer Communication
	

Fig 3.1 Communication in Peer to peer and Client/ server model

P2P systems can be used to provide anonymized routing of network traffic, massive parallel computing environments, distributed storage and other functions. Most P2P programs are focused on media sharing and P2P is therefore often associated with software piracy and copyright violation.

Features of Peer to Peer Systems

- Large scale sharing of data and resources
- No need for centralized management
- Their design of P2P system must be in such a manner that each user contributes resources to the entire system.
- All the nodes in apeer-to-peer system have the same functional capabilities and responsibilities.
- The operation of P2P system does not depend on the centralized management.
- The choice of an algorithm for theplacement of data across many hosts and the access of the data must balance the workload and ensure the availability without much overhead.

Advantages of P2P systems over client/ server architecture

1) It is easy to install and so is the configuration of computers on the network.
2) All the resources and contents are shared by all the peers, unlike server-client architecture where Server shares all the contents and resources.
3) P2P is more reliable as central dependency is eliminated. Failure of one peer doesn‟t affect the functioning of other peers. In case of Client –Server network, if server goes down whole network gets affected.
4) There is no need for full-time System Administrator. Every user is the administrator of his machine. User can control their shared resources.
5) The over-all cost of building and maintaining this type of network is comparatively very less.

Disadvantages of P2P systems over client/ server architecture

1) In this network, the whole system is decentralized thus it is difficult to administer. That is one person cannot determine the whole accessibility setting of whole network.
2) Security in this system is very less viruses, spywares,trojans, etc malwares can easily transmitted this architecture.
3) Data recovery or backup is very difficult. Each computer should have its own back-up system
4) Lot of movies, music and other copyrighted files are transferred using this type of file transfer. P2P is the technology used in torrents.

P2P Middleware

- Middleware is the software that manages and supports the different components of a distributed system.
- In essence, it sits in the middle of the system.
- Middleware is usually off Middleware is usually off-the-shelf rather than specially written software.
- A key problem in Peer-to-Peer applications is to provide a way for clients to access data resources efficiently.
- Peer clients need to locate and communicate with any available resource, even though resources may be widely distributed and configuration may be dynamic, constantly adding and removing resources and connections.
- The P2P middleware must possess the following characteristics:
  - Global Scalability
  - Load Balancing
  - Local Optimization
  - Adjusting to dynamic host availability
  - Security of data
  - Anonymity, deniability, and resistance to censorship

3.1.1 Routing Overlays

A routing overlay is a distributed algorithm for a middleware layer responsible for routing requests from any client to a host that holds the object to which the request is addressed.

- Any node can access any object by routing each request through a sequence of nodes, exploiting knowledge at each of theme to locate the destination object.
- Global User IDs (GUID) also known as **opaque identifiers** are used as names, but they do not contain the location information.
- A client wishing to invoke an operation on an object submits a request including the object‟s GUID to the routing overlay, which routes the request to a node at which a replica of the object resides.

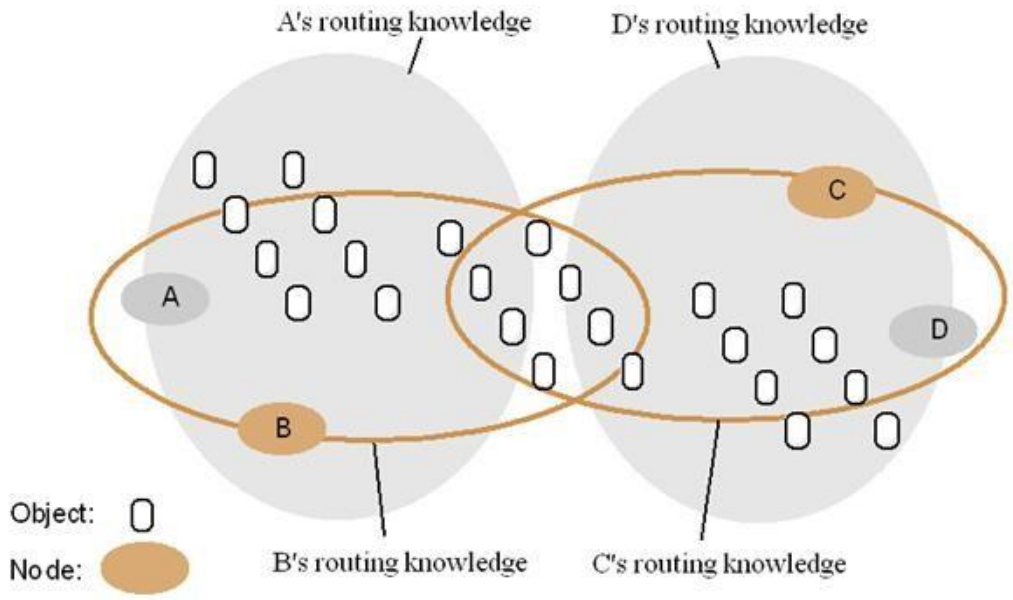


Fig 3.2: Information distribution in Routing Overlay

Differences between Overlay networks and IP routing

IP	Overlay Network
The scalability of IPV4 is limited to 2^{32} nodes and IPv6 is $2^{128.}$	Peer to peer systems can address more objects using GUID.
The load balancing is done based on topology.	Traffic patterns are independent of topology since the object locations are randomized.
Routing tables are updated asynchronously.	Routing tables are updated both synchronously and synchronously.
Failure of one node does not degrade the performance much. Redundancy is introduced in IP. n-fold replication is costly.	Routes and object references can be replicated n-fold, ensuring tolerance of n failures of nodes or connections.
Each IP can map to only one node.	Messages are routed to the nearest replica of the target object.
Secure addressing is possible only between trusted nodes.	Secure communication is possible between limited trusted systems.

Distributed Computation

- Distributed computing refers to multiple computer systems working on a single problem.
- Here a single problem is divided into many parts, and each part is solved by different computers.
- As long as the computers are networked, they can communicate with each other to solve the problem.
- The computers perform like a single entity.
- The ultimate goal of distributed computation is to maximize performance by connecting users and IT resources in a cost-effective, transparent and reliable manner.
- It also ensures fault tolerance and enables resource accessibility in the event that one of the components fails.

3.2 NAPSTER AND ITS LEGACY APPLICATION

- The Internet was originally built as a peer-to-peer system in the late 1960s to share computing resources within the US.
- Later the transfer of resources between systems took place by means of client server communication.
- Later Napster was developed for peer –to-peer file sharing especially MP3 files.
- They are not fully peer-to-peer since it used central servers to maintain lists of connected systems and the files they provided, while actual transactions were conducted directly between machines.

3.2.1 Working of Napster

- ✓ Each user must have Napster software in order to engage in file transfers.
- ✓ The user runs the Napster program. Once executed, this program checks for an Internet connection.
- ✓ If an Internet connection is detected, another connection between the user's computer and one of Napster's Central Servers will be established. This connection is made possible by the Napster file-sharing software.
- ✓ The Napster Central Server keeps a directory of all client computers connected to it and stores information on them as described above.

- ✓ If a user wants a certain file, they place a request to the Napster Centralised Server that it's connected to.
- ✓ The Napster Server looks up its directory to see if it has any matches for the user's request.
- ✓ The Server then sends the user a list of all that matches (if any) it as found including the corresponding, IP address, user name, file size, ping number, bit rate etc.
- ✓ The user chooses the file it wishes to download from the list of matches and tries to establish a direct connection with the computer upon which the desired file resides.
- ✓ It tries to make this connection by sending a message to the client computer indicating their own IP address and the file name they want to download from the client.
- ✓ If a connection is made, the client computer where the desired file resides is now considered the host.
- ✓ The host now transfers the file to the user.
- ✓ The host computer breaks the connection with the user computer when downloading is complete.

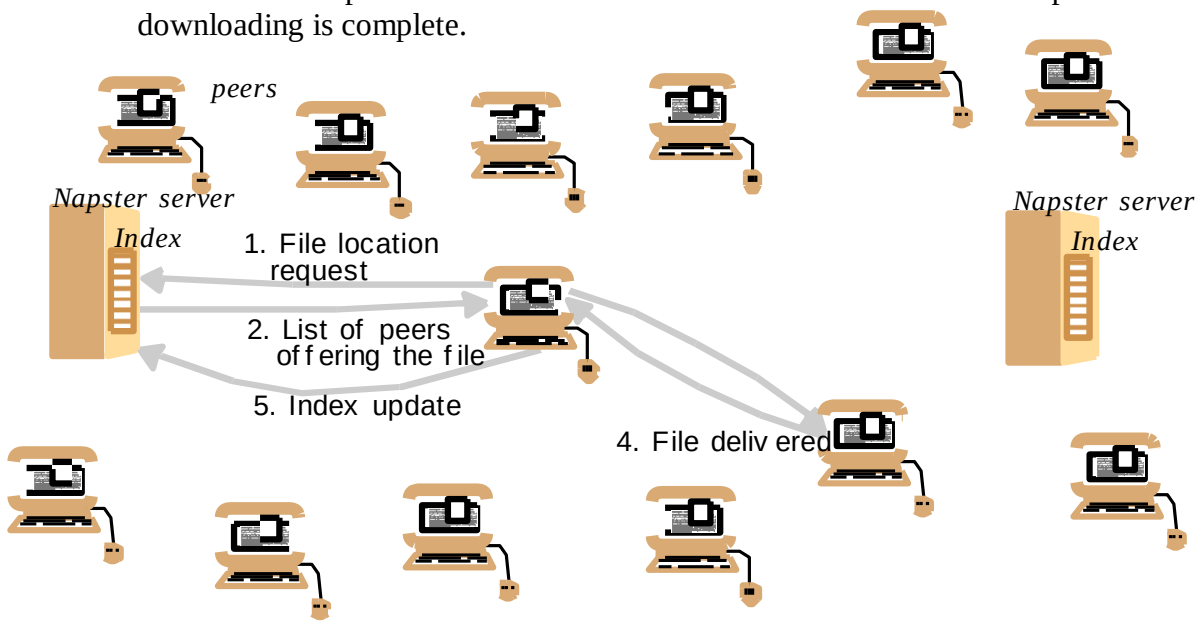


Fig 3.1: Napster

Limitations of Napster:

Discovery and addressing a file is difficult unless the location is distributed.

3.3 PEER TO PEER MIDDLEWARE

- ✓ Middleware is the software that manages and supports the different components of a distributed system.
- ✓ In essence, it sits in the middle of the system.
- ✓ Middleware is usually off Middleware is usually off-the-shelf rather than specially written software.
- ✓ A key problem in Peer-to-Peer applications is to provide a way for clients to access data resources efficiently.
- ✓ Peer clients need to locate and communicate with any available resource, even though resources may be widely distributed and configuration may be dynamic, constantly adding and removing resources and connections.
- ✓ The following are the functional requirements of the middleware:
    - ➢ Simplify construction of services across many hosts in wide network
    - ➢ Add and remove resources at will
    - ➢ Add and remove new hosts at will
    - ➢ Interface to application programmers should be simple and independent of types of distributed resources
- ✓ The following are the non functionalrequirements of the middleware:
    - ▪ Global Scalability
    - ▪ Load Balancing
    - ▪ Optimization for local interactions between neighboring peers
    - ▪ Accommodation to highly dynamic host availability
    - ▪ Security of data in an environment simplify construction of services across many hosts in wide network
    - ▪ Anonymity, deniability and resistance to censorship
- ✓ Sharing and balancing across large numbers of computers pose major design challenges to the development of middleware.
- ✓ In order to locate an object, the knowledge of object location must be distributed throughout network through replication.

3.4 ROUTING OVERLAY

A routing overlay is a distributed algorithm for a middleware layer responsible for routing requests from any client to a host that holds the object to which the request is addressed.

- Any node can access any object by routing each request through a sequence of nodes, exploiting knowledge at each of theme to locate the destination object.
- Global User IDs (GUID) also known as **opaque identifiers** are used as names, but they do not contain the location information.
- A client wishing to invoke an operation on an object submits a request including the object‟s GUID to the routing overlay, which routes the request to a node at which a replica of the object resides.
- They are actually sub-systems within the peer-to-peer middleware that are meant locating nodes and objects.
- They implement a routing mechanism in the application layer.
- They ensure that any node can access any object by routing each request thru a sequence of nodes

Features of GUID:

- ✓ They are pure names or opaque identifiers that do not reveal anything about the locations of the objects.
- ✓ They are the building blocks for routing overlays.
- ✓ They are computed from all or part of the state of the object using a function that deliver a value that is very likely to be unique. Uniqueness is then checked against all other GUIDs.
- ✓ They are not human understandable.

Process of Routing Overlay

- ∗ Client submits a request including the object GUID, routing overlay routes the request to a node at which a replica of the object resides.
- ∗ A node introduces a new object by computing its GUID and announces it to the routing overlay.
- ∗ Note that clients can remove an object and also nodes may join and leave the service

Types of Routing Overlays

1. DHT – Distributed Hash Tables. GUIDs are stored based on hash values.
2. DOLR – Distributed Object Location and Routing. DOLR is a layer over the DHT that maps GUIDs and address of nodes. GUIDs host address is notified using the Publish() operation.

3.5 PASTRY

Pastry is a generic, scalable and efficient substrate for peer-to-peer applications. Pastry nodes form a decentralized, self-organizing and fault-tolerant overlay network within the Internet.

- Pastry provides efficient request routing, deterministic object location, and load balancing in an application-independent manner.
- Furthermore, Pastry provides mechanisms that support and facilitate application-specific object replication, caching, and fault recovery.
- Each node in the Pastry network has a unique, uniform random identifier (nodeId) in a circular 128-bit identifier space.
- When presented with a message and a numeric 128-bit key, a Pastry node efficiently routes the message to the node with a nodeId that is numerically closest to the key, among all currently live Pastry nodes.
- The expected number of forwarding steps in the Pastry overlay network is O(log N), while the size of the routing table maintained in each Pastry node is only O(log N) in size (where N is the number of live Pastry nodes in the overlay network).
- At each Pastry node along the route that a message takes, the application is notified and may perform application-specific computations related to the message.
- Each Pastry node keeps track of its L immediateneighbors in the nodeId space called the **leaf set**, and notifies applications of new node arrivals, node failures and node recoveries within the leaf set.
- Pastry takes into account locality (proximity) in the underlying Internet.
- It seeks to minimize the distance messages travel, according to a scalar proximity metric like the ping delay.
- Pastry is completely decentralized, scalable, and self-organizing; it automatically adapts to the arrival, departure and failure of nodes.

Capabilities of Pastry

- **Mapping application objects to Pastry nodes**
  - Application-specific objects are assigned unique, uniform random identifiers (objIds) and mapped to the k, (k >= 1) nodes with nodeIds numerically closest to the objId.
  - The number k reflects the application's desired degree of replication for the object.
- **Inserting objects**
  - Application-specific objects can be inserted by routing a Pastry message, using the objId as the key.
  - When the message reaches a node with one of the k closest nodeIds to the objId, that node replicates the object among the other k-1 nodes with closest nodeIds (which are, by definition, in the same leaf set for k <= L/2).
- **Accessing objects**
  - Application-specific objects can be looked up, contacted, or retrieved by routing a Pastry message, using the objId as the key.
  - By definition, the message is guaranteed to reach a node that maintains a replica of the requested object unless all k nodes with nodeIds closest to the objId have failed.
- **Availability and persistence**
  - Applications interested in availability and persistence of application-specific objects maintain the following invariant as nodes join, fail and recover: object replicas are maintained on the k nodes with numerically closest nodeIds to the objId, for k> 1.
  - The fact that Pastry maintains leaf sets and notifies applications of changes in the set's membership simplifies the task of maintaining this invariant.
- **Diversity**
  - The assignment of nodeIds is uniform random, and cannot be corrupted by an attacker. Thus, with high probability, nodes with adjacent nodeIds are diverse in geographic location, ownership, jurisdiction, network attachment, etc.
  - The probability that such a set of nodes is conspiring or suffers from correlated failures is low even for modest set sizes.
  - This minimizes the probability of a simultaneous failure of all k nodes that maintain an object replica.

- **Load balancing**

  - Both nodeIds and objIds are randomly assigned and uniformly distributed in the 128-bit Pastry identifier space.
  - Without requiring any global coordination, this results in a good first-order balance of storage requirements and query load among the Pastry nodes, as well as network load in the underlying Internet.

- **Object caching**

  - Applications can cache objects on the Pastry nodes encountered along the paths taken by insert and lookup messages.
  - Subsequent lookup requests whose paths intersect are served the cached copy.
  - Pastry's network locality properties make it likely that messages routed with the same key from nearby nodes converge early, thus lookups are likely to intercept nearby cached objects.
  - This distributed caching offloads the k nodes that hold the primary replicas of an object, and it minimizes client delays and network traffic by dynamically caching copies near interested clients.

- **Efficient, scalable information dissemination**

  - Applications can perform efficient multicast using reverse path forwarding along the tree formed by the routes from clients to the node with nodeId numerically closest to a given objId.
  - Pastry's network locality properties ensure that the resulting multicast trees are efficient; i.e., they result in efficient data delivery and resource usage in the underlying Internet.

3.5.1 Pastry's Routing Algorithm

The routing algorithm involves the use of a routing table at each node to route messages efficiently. This is divided into two stages:

- **First Stage:**

  - In this stage a routing scheme is used, that routes messages correctly but inefficiently without a routing table.
  - Each active node stores a leaf set – a vector L (of size 2l) containing the GUIDs and IP addresses of the nodes whose GUIDs are numerically closest on either side of its own (l above and l below).

- Leaf sets are maintained by Pastry as nodes join and leave. Even after a node failure, they will be corrected within a short time.
- In the Pastry system that the leaf sets reflect a recent state of the system and that they converge on the current state in the face of failures up to some maximum rate of failure.
- The GUID space is treated in circular fashion.
- GUID 0‟s lower neighbour is 2128–1.
- Every leaf set includes the GUIDs and IP addresses of the current node‟s immediateneighbours.
- A Pastry system with correct leaf sets of size at least 2 can route messages to any GUID trivially as follows: any node A that receives a messageM with destination address D routes the message by comparing D with its own GUIDA and with each of the GUIDs in its leaf set and forwarding M to the node amongstthem that is numerically closest to D.

➢ **Second Stage:**

- This has a full routing algorithm, which routes a request to any node in O(log N) messages.
- Each Pastry node maintains a tree-structured routing table with GUIDs and IP addresses for a set of nodes spread throughout the entire range of 2^{128} possible GUID values, with increased density of coverage for GUIDs numerically close to its own.
- The structure of routing table is: GUIDs are viewed as hexadecimal values and the table classifies GUIDs based on their hexadecimal prefixes.
- The table has as many rows asthere are hexadecimal di gits in a GUID, so for the prototype Pastry system that weare describing, there are 128/4 = 32 rows.
- Any row n contains 15 entries – one foreach possible value of the nth hexadecimal digit, excluding the value in the local node ‟s GUID.
- Each entry in the table points to one of the potentially many nodes whose GUIDs have the relevant prefix.
- The routing process at any node A uses the information in its routing table R and leaf set L to handle each request from an application and each incoming messagefrom another node according to the algorithm.

Pastry's Routing Algorithm

To handle a message M addressed to a node D (where R[p,i] is the element at column i, row p of the routing table):

1. If (L-l < D <Ll) { // the destination is within the leaf set or is the current node.
2. Forward M to the element Li of the leaf set with GUID closest to D or the current node A.
3. } else { // use the routing table to despatch M to a node with a closer GUID
4. Find p, the length of the longest common prefix ofD and A,.and i, the (p+1)th hexadecimal digit of D.
5. If (R[p,i] ⬬ null) forward M to R[p,i] // route M to a node with a longer common prefix.
6. else { // there is no entry in the routing table.
7. Forward M to any node in L or R with a common prefix of length p but a GUID that is numerically closer.

}

}

Locality

- The Pastry routing structure is redundant.
- Each row in a routin g table contains 16 entries.
- The entries in the ithrow give the addresses of 16 nodes with GUIDs with i–1 initial hexadecimal digits thatmatch the current node "s GUID and an ith di git that takes each of the possible hexadecimal mal values.
- A well-populated Pastry overlay will contain many more nodesthan can be contained in an individual routing table; whenever a new routing table isbeing constructed a choice is made for each position between several candidates based on a proximity neighbour selection algorithm.
- A locality metric is used to compare candidates and the closest available node is chosen.
- Since the information available is not comprehensive, this mechanism cannot produce globally optimal routings, but simulations have shown that it results in routes that are on average only about 30–50% longer than the optimum.

Fault tolerance

- The Pastry routing algorithm assumes that allentries in routing tables and leaf sets refer to live, correctly functioning nodes.
- All nodessend „heartbeat" messages i.e., messages sent at fixed time intervals to indicate that thesender is alive to neighbouring nodes in their leaf sets, but information about failednodes detected in this manner may not be disseminated sufficiently rapidly to eliminaterouting errors.
- Nor does it account for malicious nodes that may attempt to interfere with correct routing.
- To overcome these problems, clients that depend upon reliable messagedelivery are expected to employ an at-least-once delivery mechanism and repeat their requests several times in the absence of a response.
- This willallow Pastry a longer time window to detect and repair node failures.
- To deal with any remaining failures or malicious nodes, a small degree of randomness is introduced into the route selection algorithm.

Dependability

- Dependability measures include the use of acknowledgements at each hop in the routing algorithm.
- If the sending host does not receive an acknowledgement after aspecified timeout, it selects an alternative route and retransmits the message.
- The nodethat failed to send an acknowledgement is then noted as a suspected failure.
- To detect failed nodes each Pastry node periodically sends aheartbeat message to its immediate neighbour to the left (i.e., with a lower GUID) in the leaf set.
- Each node also records the time of the last heartbeat message received from its immediate neighbour on the right (with a higher GUID).
- If the interval since the lastheartbeat exceeds a timeout threshold, the detecting node starts a repair procedure thatinvolves contacting the remaining nodes in the leaf set with a notification about thefailed node and a request for suggested replacements.
- Even in the case of multiplesimultaneous failures, this procedure terminates with all nodes on the left side of thefailed node having leaf sets that contain the l live nodes with the closest GUIDs.

- Suspectedfailed nodes in routing tables are probed in a similar manner to that used for the leaf setand if they fail to respond, their routing table entries are replaced with a suitablealternative obtained from a nearby node.
- A simple gossip protocol is used to periodically exchange routing table information betweennodes in order to repair failed entries and prevent slow deterioration of the localityproperties.
- The gossip protocol is run about every 20 minutes.

3.6 TAPESTRY

- Tapestry is a decentralized distributed system.
- It is an overlay network that implements simple key-based routing.
- Each node serves as both an object store and a router that applications can contact to obtain objects.
- In a Tapestry network, objects are "published" at nodes, and once an object has been successfully published, it is possible for any other node in the network to find the location at which that objects is published.
- Identifiers are either NodeIds, which refer to computers that perform routing operations, or GUIDs, which refer to the objects.
- For any resource with GUID G there is a unique root node with GUID RG that is numerically closest to G.
- Hosts H holding replicas of G periodically invoke publish(G) to ensure that newly arrived hosts become aware of the existence of G.

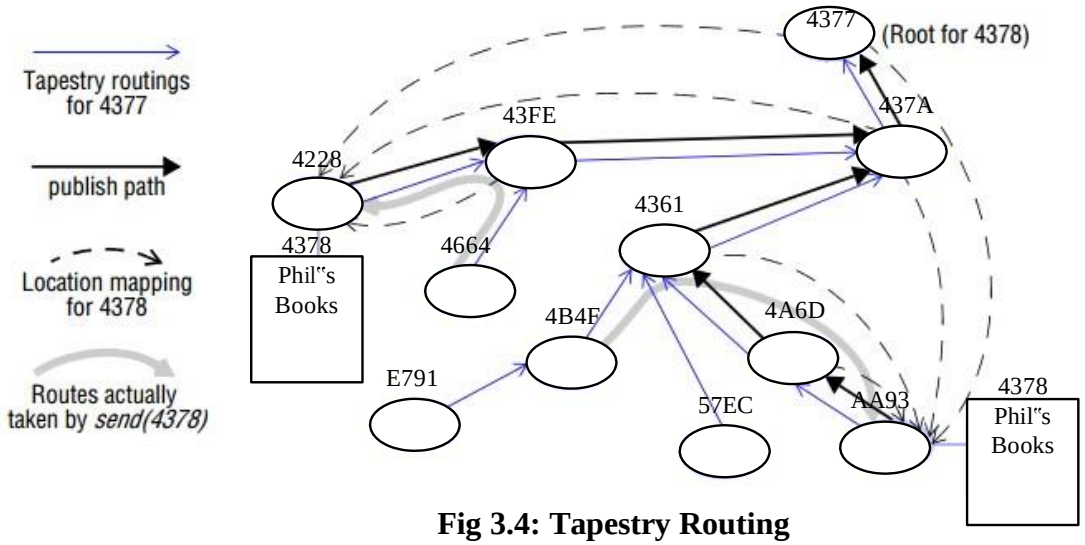


Fig 3.4: Tapestry Routing

Unstructured peer-to-peer Networks

- ✓ In structured approaches, there is an overall global policy governing the topology of the network, the placement of objects in the network and the routing or search functions used to locate objects in the network.
- ✓ There is a specific data structure underpinning the associated overlay and a set of algorithms operating over that data structure.
- ✓ These algorithms are efficient and time bounded.
- ✓ In unstructured approaches, there is no overall control over the topology or the placement of objects within the network.
- ✓ The overlay iscreated in an ad hoc manner, with each node that joins the network following simple, local rules to establish connectivity.
- ✓ In particular, a joining node will establishcontact with a set of neighbours knowing that the neighbours will also be connected tofurtherneighbours and so on, forming a network that is fundamentally decentralized andself-organizing g and hence resilient to node failure.
- ✓ To locate a given object, it is thennecessary to carry out a search of the resultant network topology.
- ✓ This approachcannot offer guarantees of being able to find the object and performance will beunpredictable.
- ✓ There is a real risk of generating excessive message traffic tolocate objects.

Differences between structured and unstructured networks

Structured Network	Unstructured Network
This guarantees to locate objects and can offer time and complexity bounds on this operation. So it has relatively low message overhead.	This is self-organizing and resilient to node failure.
This needs complexoverlay structures, which can bedifficult and costly to achieve, especially in highly dynamic environments.	Probabilistic and hence cannot offer absolute guarantees on locating objects; prone to excessive messaging overhead which can affect scalability

Gnutella

- ✓ The Gnutella network is a fully decentralized, peer-to-peer application layer network that facilitates file sharing; and is built around an open protocol developed to enable host discovery, distributed search, and file transfer.

- ✓ It consists of the collection of Internet connected hosts on which Gnutella protocol enabled applications are running.
- ✓ The Gnutella protocol makes possible all host-to-host communication through the use of messages.

GUID	Type	TTL	Hops	Payload Size
16 bytes	1 byte	1 byte	1 byte	4 bytes
23 bytes				

Fig : 3.5: Message Format

- ✓ A message consists of the following five fields:
  - The GUID field provides a unique identifier for a message on the network; the Type field indicates which type of message is being communicated.
  - TTL field enumerates the maximum number of hops that this message is allowed to traverse.
  - Hops field provides a count of the hops already traversed.
  - Payload Size field provides a byte count of all data expected to follow the message.
  - A host communicates with its peers by receiving, processing, and forwarding messages, but to do so it must follow a set of rules which help to ensure reasonable message lifetimes.
- ✓ The rules are as follows:
  1. Prior to forwarding a message, a host will decrement its TTL field and increment its Hops field. If the TTL field is found to be zero following this action, the message is not forwarded.
  2. If a host encounters a message with the same GUID and Type fields as a message already forwarded, the new message is treated as a duplicate and is not forwarded a second time.

Types of Message

There are only five types of messages

- ✦ Ping
- ✦ Pong
- ✦ Query

- ✦ Query-Hit
- ✦ Push

Ping and Pong messages facilitate host discovery, Query and Query-Hit messages make possible searching the network, and Push messages ease file transfer from firewalled hosts.Because there is no central index providing a list of hosts connected to the network, a disconnected host must have offline knowledge of a connected host in order to connect to the network.

- ✓ Once connected to the network, a host is always involved in discovering hosts, establishing connections, and propagating messages that it receives from its peers, but it may also initiate any of the following six voluntary activities: searching for content, responding to searches, retrieving content, and distributing content. A host will typically engage in these activities simultaneously.
- ✓ A host will search for other hosts and establish connections so as to satisfy a maximum requirement for active connections as specified by the user, or to replace active connections dropped by it or its peer.
- ✓ Consequently, a host tends to always maintain the maximum requirement for active connections as specified by the user, or the one connection that it needs to remain connected to the network.
- ✓ To engage in host discovery, a host must issue a Ping message to the host of which it has offline knowledge.
- ✓ That host will then forward the ping message across its open connections, and optionally respond to it with a Pong message.
- ✓ Each host that subsequently receives the Ping message will act in a similar manner until the TTL of the message has been exhausted.
- ✓ A Pong message may only be routed along the reverse of the path that carried the Ping message to which it is responding.
- ✓ After having discovered a number of other hosts, the host that issued the initial ping message may begin to open further connections to the network.
- ✓ Doing so allows the host to issue ping messages to a wider range of hosts and therefore discover a wider range of other hosts, as well as to begin querying the network.
- ✓ In searching for content, a host propagates search queries across its active connections.
- ✓ Those queries are then processed and forwarded by its peers.

- ✓ When processing a query, a host will typically apply the query to its local database of content, and respond with a set of URLs pointing to the matching files.
- ✓ The propagation of Query and Query-Hit messages is identical to that of Ping and Pong messages.
- ✓ A host issues a Query message to the hosts to which it is connected.
- ✓ The hosts receiving that Query message will then forward it across their open connections, and optionally respond with a Query-Hit message.
- ✓ Each host that subsequently receives the Query message will act in a similar manner until the TTL of the message has been exhausted.
- ✓ Query-Hit messages may only be routed along the reverse of the path that carried the Query message to which it is a response.
- ✓ The sharing of content on the Gnutella network is accomplished through the use of the HTTP protocol.
- ✓ Due to the decentralized nature of the architecture, each host participating in the Gnutella network plays a key role in the organization and operation of the network.
- ✓ Both the sharing of host information, as well as the propagation of search requests and responses are the responsibility of all hosts on the network rather than a central index.

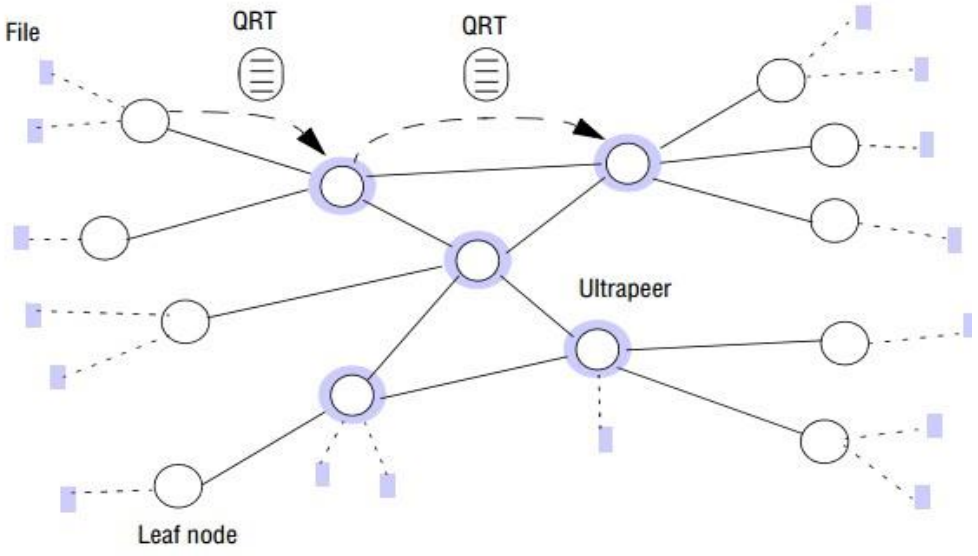


Fig 3.6: Elements in Gnutella Network

- ✓ The Gnutella network architecture avoids creating the single-point-of-failure that, theoretically, threatens other networks.

- ✓ The Gnutella network exhibits a great deal of fault-tolerance, continuing to function even as subsets of hosts become unavailable.
- ✓ Each host in the Gnutella network is capable of acting as both a server and client, allowing a user to distribute and retrieve content simultaneously.
- ✓ This spreads the provision of content across many hosts, and helps to eliminate the bottlenecks that typically occur when placing enormous loads on a single host.
- ✓ In this network, the amount and variety of content available to a user scales with the number of users participating in the network.

3.7 DISTRIBUTED FILE SYSTEMS

- ✓ The purpose of a distributed file system (DFS) is to allow users of physically distributed computers to share data and storage resources by using a common file system.
- ✓ A typical configuration for a DFS is a collection of workstations and mainframes connected by a local area network (LAN).
- ✓ A DFS is implemented as part of the operating system of each of the connected computers.

Distributed file systems support the sharing of information in the form of files and hardware resources.

- ✓ The following are the modules of a file systems:
  - ∗ Directory module: Relates file names to the disk.
  - ∗ File module: Relates file ID to the files
  - ∗ Access Control module: Manages access rights
  - ∗ File access module: Controls read or write operation on the files
  - ∗ Block module: Allocates disk blocks and helps in accessing them
  - ∗ Device module: Manages disk I/O and buffering.

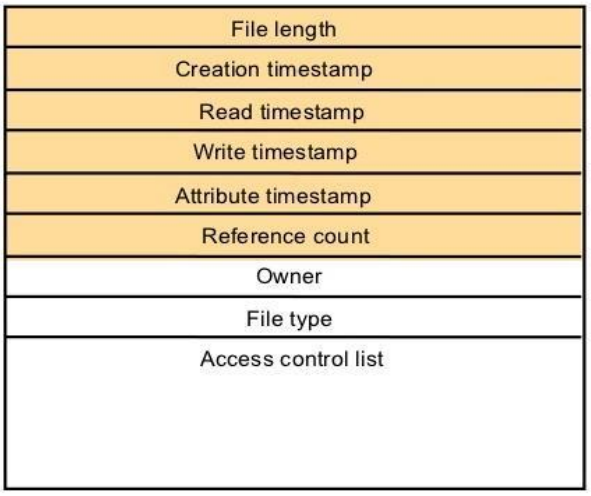


Fig 3.6: File Attribute Record Structure

File Attribute Record Structure

- File length specifies the size of the file.
- Creation Timestamp is the time at which the file was created.
- Read Timestamp is the time at which the last read operation was made.
- Write Timestamp is the time at which the last write operation was made.
- Reference Count is the number or links to the file.
- Owner is the creator of the file who has full access rights.
- File type describes the content of the file.
- Access Control List contains the list of users and the access rights given to them.

3.7.1 Unix File Systems

The time sharing Unix file system is a model for the distributed file systems. The following some of the unix commands to access files:

- filedes= open(name, mode): Opens an already existing file.
- filedes=creat(name, mode): Creates a new file
- status= close(filedes): Closes a file.
- count= read(filedes, buffer, n): Transfers n bytes from the filedes to buffer and returns the number of bytes transferred.
- count= write(filedes, buffer, n): Transfers n bytes from the filedes to buffer and returns the number of bytes transferred. This advances the write file pointer.

- pos= lseek(filedes, offset, whence): Moves the read-write pointer to offset depending on whence.
- status=unlink(name): renames the specified file.
- status=unlink(name1, name2): Adds a new name (name2) for a file (name1).
- status= stat(name, buffer): Gets the file attributes for file into the buffer.

3.7.2 Characteristics of a File system

- ✓ File systems are responsible for the organization, storage, retrieval, naming, sharing and protection of files.
- ✓ Data and attributes are the primary characteristics of a file system.
- ✓ The data consist of a sequence of data items .
- ✓ 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.
- ✓ A directory is a special file that provides a mapping from text names to internal file identifiers.
- ✓ The term metadata is 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.
- ✓ The file system also maintains the access control lists.

3.7.3 Requirements of distributed file systems

Transparency is operating in such a way as to not be perceived by users. The distributed file system demands the following transparency requirements:

- **Login Transparency:** User can log in at any host with uniform login procedure and perceive a uniform view of the file system.
- **Access Transparency:** Client process on a host has uniform mechanism to access all files in system regardless of files are on local/remote host.
- **Location Transparency:** The names of the files do not reveal their physical location.
- **Concurrency Transparency:** An update to a file should not have effect on the correct execution of other process that is concurrently sharing a file.
- **Replication Transparency:** Files may be replicated to provide redundancy for availability and also to permit concurrent access for efficiency.

Apart from the above transparency properties, the file systems also need the following:

- **Fault Tolerance:** It is a design that enables a system to continue operation, possibly at a reduced level, rather than failing completely, when some part of the system fails.
- **Network Transparency:** Same access operation as if they are local files.
- **Location Independence:** The file name should not be changed when the physical location of the file changes.
- **User Mobility:** User should be able to access the file from any where.
- **File Mobility:** Moves files from one place to the other in a running system.

Examples:

- ✓ **The GOOGLE File System:** A scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.
- ✓ **The CODA distributed file system:** Developed at CMU, incorporates many distinguished features which are not present in any other distributed file system.

3.8 FILE SERVICE ARCHITECTURE

File service architecture 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
- A client module.

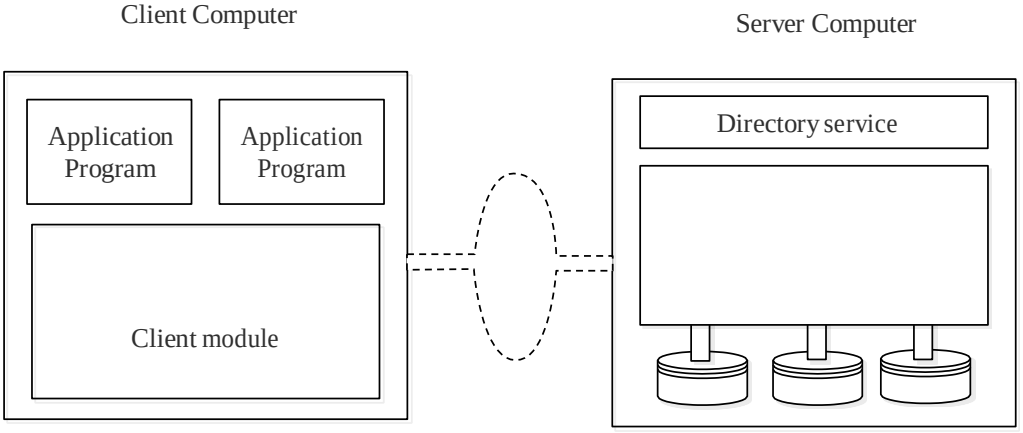


Fig 3.9: File Service Architecture

Flat file service:

- ✓ This is concerned with the implementation of operations on the contents of file.
- ✓ Unique File Identifiers (UFIDs) are used to refer to files in all requests for flat file service operations.
- ✓ UFIDs are long sequences of bits chosen so that each file has a unique among all of the files in a distributed system.

Directory service:

- ✓ This provides mapping between text names for the files and their UFIDs.
- ✓ Clients may obtain the UFID of a file by quoting its text name to directory service.
- ✓ Directory service supports functions needed generate directories, to add new files to directories.

Client module:

- ✓ It runs on each computer and provides integrated service (flat file and directory) as a single API to application programs.
- ✓ It holds information about the network locations of flat-file and directory server processes; and achieve better performance through implementation of a cache of recently used file blocks at the client.

The following are the operations in file service:

- Read(FileId, i, n)→Data :Reads a sequence of up to n items from a file starting at item i and returns it in Data.
- Write(FileId, i, Data):Write a sequence of Data to a file, starting at item i, extending the file if necessary.
- Create() →FileId: Creates a new file of length0 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 .

➢ **Access control in files:**

In distributed implementations, access rights checks have to be performed at the server because the server RPC interface is an otherwise unprotected point of access to files.

- **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.

- **File Group**

  - ✓ A file group is a collection of files that can be located on any server or moved between servers while maintaining the same names.
  - ✓ A similar construct is used in a UNIX file system.
  - ✓ It helps with distributing the load of file serving between several servers.
  - ✓ File groups have identifiers which are unique throughout the system.
  - ✓ To construct a globally unique ID we use some unique attribute of the machine on which it is created, e.g. IP number, even though the file group may move subsequently.

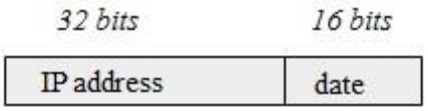


Fig 3.9: File group ID

3.9 ANDREW FILE SYSTEM (AFS)

- This was developed by Carnegie Mellon University as part of Andrew distributed computing environments (in 1986).
- The public domain implementation is available on Linux (LinuxAFS)
- It was adopted as a basis for the DCE/DFS file system in the Open Software Foundation (OSF, www.opengroup.org) DEC (Distributed Computing Environment).
- Like NFS, AFS provides transparent access to remote shared files for UNIX programs running on workstations.
- AFS is implemented as two software components that exist at UNIX processes called Vice and Venus.
- The files available to user processes running on workstations are either local or shared.
- Local files are handled as normal UNIX files.
- They are stored on the workstation‟s disk and are available only to local user processes.

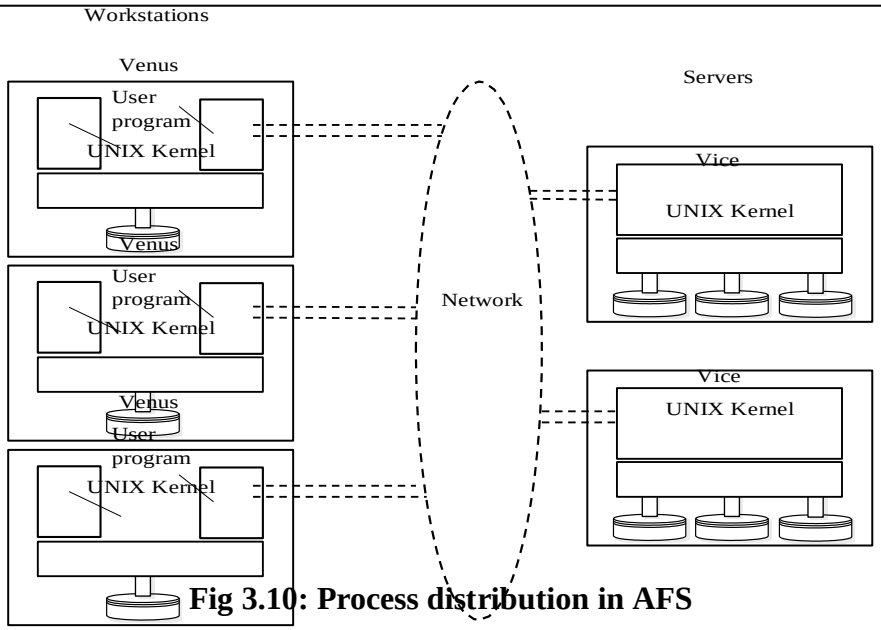


Fig 3.10: Process distribution in AFS

- Shared files are stored on servers, and copies of them are cached on the local disks of workstations.
- The UNIX kernel in each workstation and server is a modified version of BSD UNIX.
- The modifications are designed to intercept open, close and some other file system calls when they refer to files in the shared name space and pass them to the Venus process in the client computer.

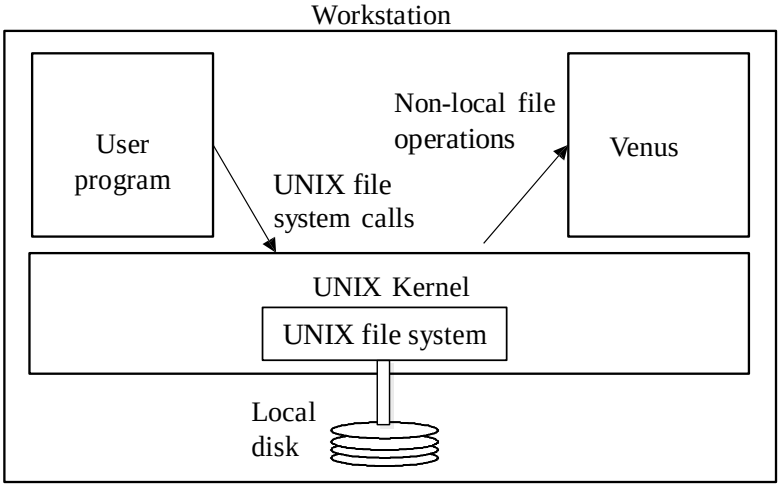


Fig 3.11: System call interception

Components of Vice service interface

- Fetch(fid)→attr, data: Returns the attributes (status) and, optionally, the contents of
  the file identified by fid and records a callback promise on it.
- Store(fid, attr, data): Updates the attributes and (optionally) the contents of a specified file.
- Create() →fid: Creates a new file and records a callback promise on it.
- Remove(fid) Deletes the specified file.
- SetLock(fid, mode): Sets a lock on the specified file or directory. The mode of the lock may be shared or exclusive. Locks that are not removed expire after 30 minutes.
- ReleaseLock(fid):Unlocks the specified file or directory.
- RemoveCallback(fid): Informs the server that a Venus process has flushed a file from
  its cache.
- BreakCallback(fid:) Call made by a Vice server to a Venus process; cancels the call back promise on the relevant file.

3.10 FILE ACCESSING MODELS

File access models are methods used for accessing remote files and the unit of data access. A distributed file system may use one of the following models to service a client‟s file access request when the accessed file is remote:

1. **Remote service model**
   - Processing of a client‟s request is performed at the server‟s node.
   - Thus, the client‟s request for file access is delivered across the network as a message to the server, the server machine performs the access request, and the result is sent to the client.
   - This need to minimize the number of messages sent and the overhead per message.
2. **Data-caching model**
   - This model attempts to reduce the network traffic of the previous model by caching the data obtained from the server node.
   - This takes advantage of the locality feature of the found in file accesses.

- ❖ A replacement policy such as LRU is used to keep the cache size bounded.
- ❖ This model reduces network traffic it has to deal with the cache coherency problem during writes, because the local cached copy of the data needs to be updated, the original file at the server node needs to be updated and copies in any other caches need to be updated.
- ❖ The data-caching model offers the possibility of increased performance and greater system scalability because it reduces network traffic, contention for the network, and contention for the file servers. Hence almost all distributed file systems implement some form of caching.
- ❖ In file systems that use the data-caching model, an important design issue is to decide the **unit of data transfer.**
- ❖ This refers to the fraction of a file that is transferred to and form clients as a result of single read or write operation.

3. File-level transfer model

- ❖ In this model when file data is to be transferred, the entire file is moved.
- ❖ File needs to be transferred only once in response to client request and hence is more efficient than transferring page by page which requires more network protocol overhead.
- ❖ This reduces server load and network traffic since it accesses the server only once. This has better scalability.
- ❖ Once the entire file is cached at the client site, it is immune to server and network failures.
- ❖ This model requires sufficient storage space on the client machine.
- ❖ This approach fails for very large files, especially when the client runs on a diskless workstation.
- ❖ If only a small fraction of a file is needed, moving the entire file is wasteful.

4. Block-level transfer model

- ❖ File transfer takes place in file blocks.
- ❖ A file block is a contiguous portion of a file and is of fixed length (can also be a equal to a virtual memory page size).
- ❖ This does not require client nodes to have large storage space.
- ❖ It eliminates the need to copy an entire file when only a small portion of the data is needed.

- ❖ When an entire file is to be accessed, multiple server requests are needed, resulting in more network traffic and more network protocol overhead. NFS uses block-level transfer model.

5. Byte-level transfer model

- ❖ Unit of transfer is a byte.
- ❖ Model provides maximum flexibility because it allows storage and retrieval of an arbitrary amount of a file, specified by an offset within a file and length.
- ❖ The drawback is that cache management is harder due to the variable-length data for different access requests.

6. Record-level transfer model

- ❖ This model is used with structured files and the unit of transfer is the record.

3.11 FILE-SHARING SEMANTICS

- ✓ Multiple users may access a shared file simultaneously.
- ✓ An important design issue for any file system is to define when modifications of file data made by a user are observable by other users.
- ✓ The UNIX semantics is implemented in file systems for single CPU systems because it is the most desirable semantics and because it is easy to serialize all read/write requests.
- ✓ Implementing UNIX semantics in a distributed file system is not easy.
- ✓ One may think that this can be achieved in a distributed system by disallowing files to be cached at client nodes and allowing a shared file to be managed by only one file server that processes all read and write requests for the file strictly in the order in which it receives them.
- ✓ However, even with this approach, there is a possibility that, due to network delays, client requests from different nodes may arrive and get processed at the server node in an order different from the actual order in which the requests were made.
- ✓ Also, having all file access requests processed by a single server and disallowing caching on client nodes is not desirable in practice due to poor performance, poor scalability, and poor reliability of the distributed file system.
- ✓ Hence distributed file systems implement a more relaxed semantics of file sharing.
- ✓ Applications that need to guarantee UNIX semantics should provide mechanisms (e.g. mutex lock etc) themselves and not rely on the underlying semantics of sharing provided by the file system.

3.12 NAMING IN DISTRIBUTED SYSTEMS

- A Name is a string of bits used to refer to an entity.
- We operate on an entity through its Access Point. The Address is the name of the access point.
- The naming faciliy of a distributed operating system enables users and programs to assign character-string names to objects and subsequently use these names to refer to those objects.
- The locating faciliy, which is an integral part of the naming facility, maps an object's name to the object's location in a distributed system.
- The naming and locating facilities jointly form a naming system that provides the users with an abstraction of an object that hides the details of how and where an object is actually located in the network.
- It provides a further level of abstraction when dealing with object replicas.
- Given an object name, it returns a set of the locations of the object's replicas.
- The naming system plays a very important role in achieving the goal of
    - location transparency,
    - facilitating transparent migration
    - replication of objects, v object sharing.

Example:

- ✓ Telephone as Access Point to a person.
- ✓ The Telephone Number then becomes the address of the person.
- ✓ Person can have several telephone numbers.
- ✓ Entity can have several addresses

3.12.1 Identifiers:

- Identifiers are special names that uniquely identify an entity.
- An identifier refers to at most one entity.
- Each entity is referred to at most one identifier.
- An identifier always refers to the same entity (never reused).

3.12.2 Namespaces:

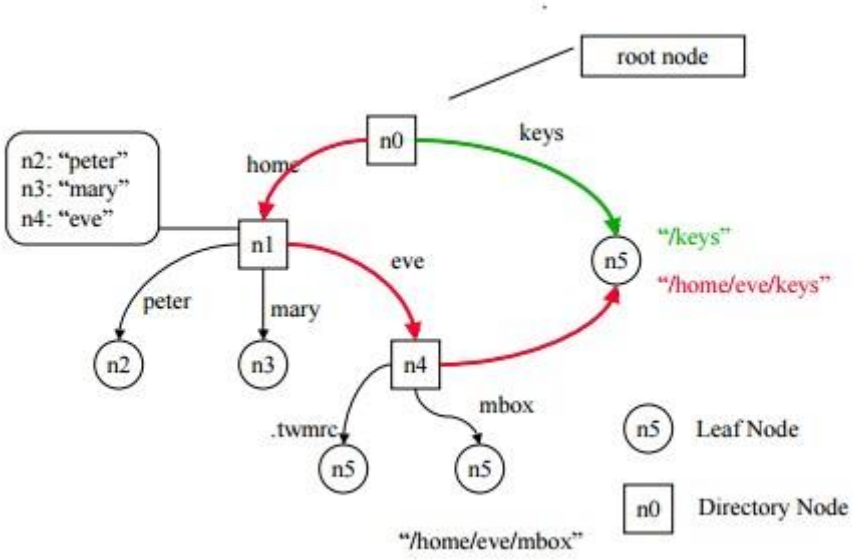


Fig 3.12: Namespaces

- Names are organized into Name Space.
- Entities in a structured name space are named by a path name.
- Leaf nodes represent named entities (e.g., files) and have only incoming edges .
- Directory nodes have named outgoing edges and define the path used to find a leaf node

Attribute based naming

- This allows a user to search for an entity whose name is not known.
- Entities are associated with various attributes, which can have specific values.
- By specifying a collection of <attribute, value> pairs, a user can identify one (or more) entities
- Attribute based naming systems are also referred to as **directory services**, as opposed to naming systems.

3.12.3 Naming resolution

It is the process of looking up a name. The path name of the file is given as

N:<label-1, label-2,…, label-n>

Name resolution is the process of mapping an object'sname to the object's properties, such as its location.

- ✓ Since an object's properties are stored and maintained by the authoritative name servers of that object, name resolution is basically the process of mapping an object's name to the authoritative name servers of that object.
- ✓ Once an authoritative name server of the object has been located, operations can be invoked to read or update the object's properties.
- ✓ Each name agent in a distributed system knows about at least one name server apriori.
- ✓ To get a name resolved, a client first contacts its name agent, which in turn contacts a known name server, which may in turn contact other name servers.

Absolute and Relative Names

- A current working context is also known by the shorter names current context or working context.
- According to this concept, a user is always associated with a context that is his or her current context.
- A user can change his or her current context whenever he or she desires.

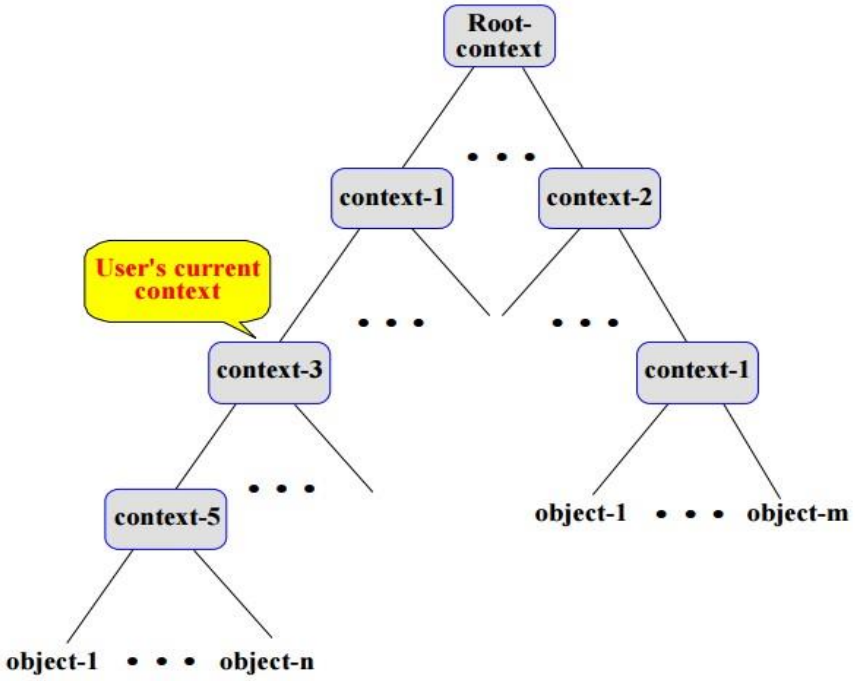


Fig 3.12: Tree structured name space

- An **absolute name** begins at the root context of the name space tree and follows a path down to the specified object, giving the context names on the path.

- On the other hand, a relative name defines a path from the current context to the specified object. It is called a relative name because it is "relative to" (start from) the user's current context.

In this method, a user may specify an object in any of the following ways:

1. Using the full (absolute) name
2. Using a relative name
3. Changing the current context first and then using a relative name

Implementing Name Spaces

Three layers used to implement such distributed name spaces

- Global layer: root node and its children
- Administrational layer: directory nodes within a single organization
- Managerial layer

Item	Global	Administrational	Managerial
Geographical scale of network	Worldwide	Organization	Department
Total number of nodes	Few	Many	Vast numbers
Responsiveness of lookups	Seconds	Milliseconds	Immediate
Updata propagation	Lazy	Immediate	Immediate
Number of replicas	Many	None or few	None
Is client-side caching applied?	Yes	Yes	Sometimes

Name Space Distribution

Iterative name resolution

- Recursive name resolution puts a higher performance demand on each name server.
- Too high for global layer name servers
- In this method, the name agent forwards the name resolution request to the name server that stores the first context needed to start the resolution of the given name.
- After this, the name servers that store the contexts of the given pathname are recursively activated one after another until the authority attribute of the named

object is extracted from the context corresponding to the last component name of the pathname.

- The last name server returns the authority attribute to its previous name server, which then returns it to its own previous name server, and so on.
- Finally, the fast name server that received the request from the name agent returns the authority attribute to the name agent

Advantages of recursive name resolution:

- Caching is more effective
- Communication costs may be reduced

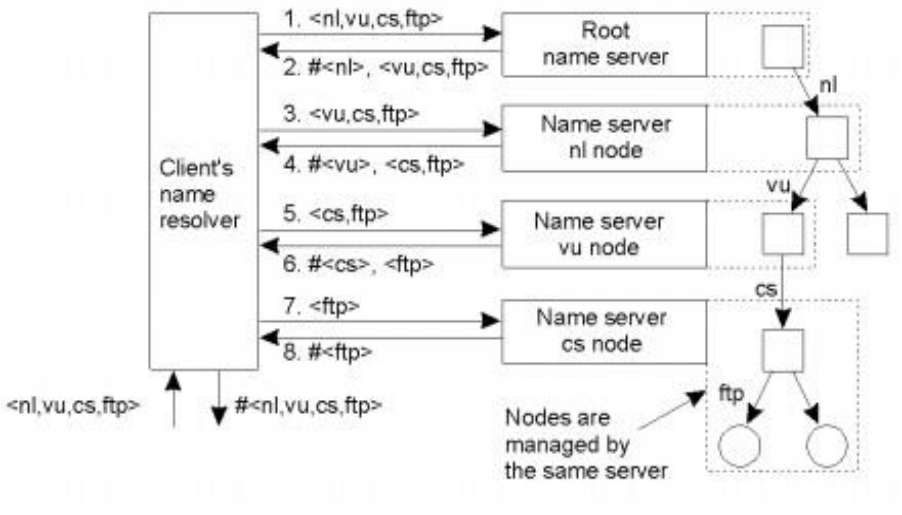


Fig 3.12: Principle of iterative name resolution

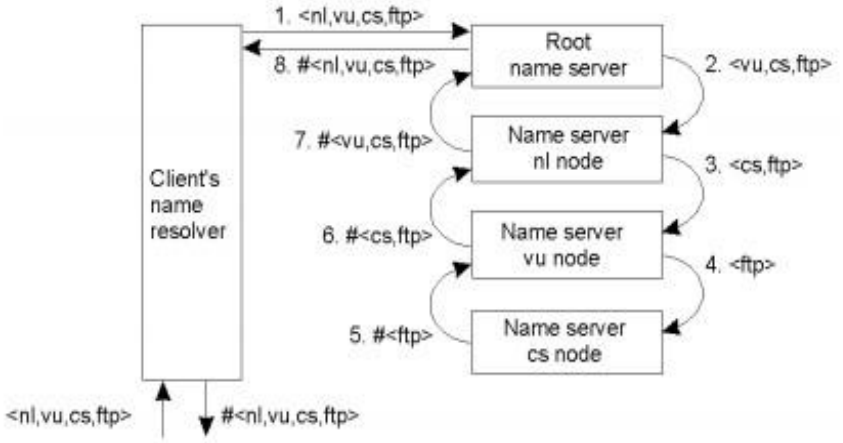


Fig 3.13: Principle of iterative name resolution

Recursive Name resolution:

- In this method, name servers do not call each other directly.
- Rather, the name agent retains control over the resolution process and one by one calls each of the servers involved in the resolution process.
- To continue the name resolution, the name agent sends a name resolution request along with the unresolved portion of the name to the next name server.
- The process continues until the name agent receives the authority attribute of the named object

3.12.4: Name Caches

Caching is an important technique for building responsive, scalable distributed systems. A cache can be maintained either by the client or the server or by both.

Types of Name caches

- **Client name caches**

  Caching at the client is an effective way of pushing the processing workload from the server out to client devices,if a client has the capacity.

- **Server based cache**

  If result data is likely to be reused by multiple clients or if the client devices do not have the capacity then caching at the server is more effective.

- **Multi-level caching**

  Caches can be maintained at multiple levels. For example, caches can be maintained at all clients and all servers. Use of a cache at one level reduces the number of requests handled at the levels below.

3.12.5: Lightweight Directory Access Protocol (LDAP)

LDAP is a standard technology for building computer network directories. A network directory is a specialized database that stores information about devices, applications, people and other aspects of a computer network.

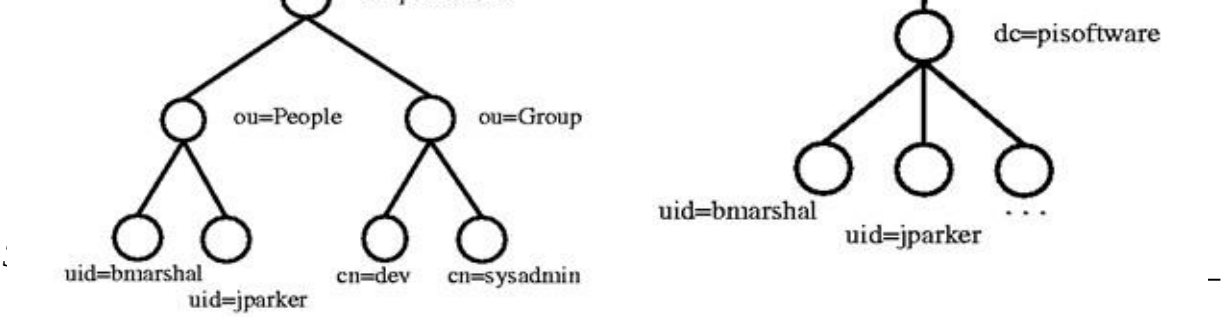


Fig 3.14: LDAP Tree structure

Working of LDAP:

- Client starts an LDAP session by connecting to an LDAP server.
- The default port on TCP is 389.
- Client sends operation requests to the server and the server sends responses in turn.
- With some exceptions the client need not wait for a response before sending the next request. Server may send the responses in any order.
- The client may request the following operations:
  - Start TLS : Optionally protect the connection with Transport Layer Security (TLS), to have a more secure connection
  - Bind - authenticate and specify LDAP protocol version
  - Search - search for and/or retrieve directory entries
  - Compare - test if a named entry contains a given attribute value
  - Add a new entry
  - Delete an entry
  - Modify an entry
  - Modify Distinguished Name (DN) - move or rename an entry
  - Abandon - abort a previous request

- Extended Operation - generic operation used to define other operations
- Unbind - close the connection (not the inverse of Bind)

– In addition the server may send "Unsolicited Notifications" that are not responses to any request, e.g. before it times out a connection.

Directory Structure

- ✓ Directory is a tree of directory entries.
- ✓ Each entry consists of a set of attributes.
- ✓ An attribute has: a name , an attribute type or attribute description and one or more values
- ✓ Attributes are defined in a schema.
- ✓ Each entry has a unique identifier called Distinguished Name (DN).
- ✓ The DN consists of its Relative Distinguished Name (RDN) constructed from some attribute(s) in the entry.
- ✓ It is followed by the parent entry's DN.
- ✓ Think of the DN as a full filename and the RDN as a relative filename in a folder.
- ✓ DN may change over the lifetime of the entry.
- ✓ To reliably and unambiguously identify entries, a UUID might be provided in the set of the entry's operational attributes.

REVIEW QUESTIONS

PART - A

1. Define peer-to-peer communications.

Peer-to-peer (P2P) is a decentralized communications model in which each party has the same capabilities and either party can initiate a communication session.

2. Give the features of Peer to Peer Systems

- Large scale sharing of data and resources
- No need for centralized management
- Their design of P2P system must be in such a manner that each user contributes resources to the entire system.
- All the nodes in apeer-to-peer system have the same functional capabilities and responsibilities.
- The operation of P2P system does not depend on the centralized management.
- The choice of an algorithm for theplacement of data across many hosts and the access of the data must balance the workload and ensure the availability without much overhead.

3. List the advantages of P2P systems over client/ server architecture

1. It is easy to install and so is the configuration of computers on the network.
2. All the resources and contents are shared by all the peers, unlike server-client architecture where Server shares all the contents and resources.
3. P2P is more reliable as central dependency is eliminated. Failure of one peer doesn‟t affect the functioning of other peers. In case of Client –Server network, if server goes down whole network gets affected.
4. There is no need for full-time System Administrator. Every user is the administrator of his machine. User can control their shared resources.
5. The over-all cost of building and maintaining this type of network is comparatively very less.

4. List the disadvantages of P2P systems over client/ server architecture

1. In this network, the whole system is decentralized thus it is difficult to administer. That is one person cannot determine the whole accessibility setting of whole network.
2. Security in this system is very less viruses, spywares,trojans, etc malwares can easily transmitted this architecture.

3. Data recovery or backup is very difficult. Each computer should have its own back-up system
4. Lot of movies, music and other copyrighted files are transferred using this type of file transfer. P2P is the technology used in torrents.

5. Give the characteristics of peer-to-peer middleware.

- The P2P middleware must possess the following characteristics:
  - Global Scalability
  - Load Balancing
  - Local Optimization
  - Adjusting to dynamic host availability
  - Security of data
  - Anonymity, deniability, and resistance to censorship

6. Define routing overlay.

A routing overlay is a distributed algorithm for a middleware layer responsible for routing requests from any client to a host that holds the object to which the request is addressed.

7. Give the differences between Overlay networks and IP routing

IP	Overlay Network
The scalability of IPV4 is limited to 2^{32} nodes and IPv6 is $2^{128.}$	Peer to peer systems can address more objects using GUID.
The load balancing is done based on topology.	Traffic patterns are independent of topology since the object locations are randomized.
Routing tables are updated asynchronously.	Routing tables are updated both synchronously and synchronously.
Failure of one node does not degrade the performance much. Redundancy is introduced in IP. n-fold replication is costly.	Routes and object references can be replicated n-fold, ensuring tolerance of n failures of nodes or connections.
Each IP can map to only one node.	Messages are routed to the nearest replica of the target object.
Secure addressing is possible only between trusted nodes.	Secure communication is possible between limited trusted systems.

8. What is Napster?

Napster was developed for peer –to-peer file sharing especially MP3 files.They are not fully peer-to-peer since it used central servers to maintain lists of connected systems and the files they provided, while actual transactions were conducted directly between machines.

9. Give the features of GUID.

- They are pure names or opaque identifiers that do not reveal anything about the locations of the objects.
- They are the building blocks for routing overlays.
- They are computed from all or part of the state of the object using a function that deliver a value that is very likely to be unique. Uniqueness is then checked against all other GUIDs.
- They are not human understandable.

10. Give the types of routing overlays.

DHT – Distributed Hash Tables. GUIDs are stored based on hash values.

DOLR – Distributed Object Location and Routing. DOLR is a layer over the DHT that maps GUIDs and address of nodes. GUIDs host address is notified using the Publish() operation.

11. **Define pastry**.

Pastry is a generic, scalable and efficient substrate for peer-to-peer applications. Pastry nodes form a decentralized, self-organizing and fault-tolerant overlay network within the Internet.

12. Give the Capabilities of Pastry

- Mapping application objects to Pastry nodes
- Inserting objects
- Accessing objects
- Availability and persistence
- Diversity
- Load balancing
- Object caching
- Efficient, scalable information dissemination

13. What is tapestry?

Tapestry is a decentralized distributed system.It is an overlay network that implements simple key-based routing. Each node serves as both an object store and a router that applications can contact to obtain objects.

14. Give the differences between structured and unstructured networks

Structured Network	Unstructured Network
This guarantees to locate objects and can offer time and complexity bounds on this operation. So it has relatively low message overhead.	This is self-organizing and resilient to node failure.
This needs complexoverlay structures, which can bedifficult and costly to achieve, especially in highly dynamic environments.	Probabilistic and hence cannot offer absolute guarantees on locating objects; prone to excessive messaging overhead which can affect scalability

15. What is Gnutella?

The Gnutella network is a fully decentralized, peer-to-peer application layer network that facilitates file sharing; and is built around an open protocol developed to enable host discovery, distributed search, and file transfer.

16. What is DFS?

The purpose of a distributed file system (DFS) is to allow users of physically distributed computers to share data and storage resources by using a common file system. A typical configuration for a DFS is a collection of workstations and mainframes connected by a local area network (LAN). Distributed file systems support the sharing of information in the form of files and hardware resources.

17. What are the requirements of distributed file systems?

Transparency is operating in such a way as to not be perceived by users. The distributed file system demands the following transparency requirements:

- Login Transparency: User can log in at any host with uniform login procedure and perceive a uniform view of the file system.
- Access Transparency: Client process on a host has uniform mechanism to access all files in system regardless of files are on local/remote host.
- Location Transparency: The names of the files do not reveal their physical location.

- Concurrency Transparency: An update to a file should not have effect on the correct execution of other process that is concurrently sharing a file.
- Replication Transparency: Files may be replicated to provide redundancy for availability and also to permit concurrent access for efficiency.

18. What are the components in file structures?

File service architecture 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
- A client module.

19. What is AFS?

This was developed by Carnegie Mellon University as part of Andrew distributed computing environments (in 1986).The public domain implementation is available on Linux (LinuxAFS). It was adopted as a basis for the DCE/DFS file system in the Open Software Foundation (OSF, www.opengroup.org) DEC (Distributed Computing Environment).Like NFS, AFS provides transparent access to remote shared files for UNIX programs running on workstations

20. Define file access modes.

File access models are methods used for accessing remote files and the unit of data access.

21. What is naming and locating facility?

The naming faciliy of a distributed operating system enables users and programs to assign character-string names to objects and subsequently use these names to refer to those objects. The locating faciliy, which is an integral part of the naming facility, maps an object's name to the object's location in a distributed system.

22. Define naming resolution.

Name resolution is the process of mapping an object'sname to the object's properties, such as its location.

23. What is absolute name?

An absolute name begins at the root context of the name space tree and follows a path down to the specified object, giving the context names on the path.

24. What is relative name?

A relative name defines a path from the current context to the specified object. It is called a relative name because it is "relative to" (start from) the user's current context.

25. What are the layers in name spaces?

Three layers used to implement such distributed name spaces

- Global layer: root node and its children
- Administrational layer: directory nodes within a single organization
- Managerial layer

26. What is LDAP?

LDAP is a standard technology for building computer network directories. A network directory is a specialized database that stores information about devices, applications, people and other aspects of a computer network.

PART – B

1. Explain about peer to peer communication.
2. Explain the working of routing overlays.
3. Describe Napster.
4. Write in detail about peer to peer middleware.
5. Explain about pastry.
6. Describe tapestry.
7. Write in detail about distributed file system.
8. Describe file service architecture.
9. Write in detail about Andrew file system.
10. Describe the file accessing models.
11. Elucidate the file sharing semantics.
12. Describe the naming in distributed systems.

4

SYNCHRONIZATION AND REPLICATION

4.1 INTRODUCTION TO SYNCHRONIZATION

Synchronizing data in a distributed system is an enormous challenge in and of itself. In single CPU systems, critical regions, mutual exclusion, and other synchronization problems are solved using methods such as semaphores. These methods will not work in distributed systems because they implicitly rely on the existence of shared memory.

Examples:

- Two processes interacting using a semaphore must both be able to access the semaphore. In a centralized system, the semaphore is stored in the kernel and accessed by the processes using system calls.
- If two events occur in a distributed system, it is difficult to determine which event occurred first.

Communication between processes in a distributed system can have unpredictable delays, processes can fail, messages may be lost synchronization in distributed systems is harder than in centralized systems because the need for distributed algorithms. The following are the properties of distributed algorithms:

- The relevant information is scattered among multiple machines.
- Processes make decisions based only on locally available information.
- A single point of failure in the system should be avoided.
- No common clock or other precise global time source exists

4.1.1 Issues in Clocks

Physical clocks in computers are realized as crystal oscillation counters at the hardware level. Clock skew(offset) is common problem with clocks. **Clock skew** is defined as the difference between the times on two clocks and **Clock drift** is the count time at different rates. **Clock drift rate** is the difference in precision between a prefect reference clock and a physical clock.

4.1.2 Synchronising clocks

There are two ways to synchronise a clock:

- **External synchronization**
  - This method synchronize the process‟s clock with an authoritative external reference clock S(t) by limiting skew to a delay bound D > 0 - |S(t) - Ci(t) | < D for all t.
  - For example, synchronization with a UTC (Coordinated Universal Time)source.
- **Internal synchronization**
  - Synchronize the local clocks within a distributed system to disagree by not more than a delay bound D > 0, without necessarily achieving external synchronization - |Ci(t) - Cj(t)| < D for all i, j, t n
  - For a system with external synchronization bound of D, the internal synchronization is bounded by 2D.

Checking the correctness of a clock

- ✓ If drift rate falls within a bound r > 0, then for any t and t‟ with t‟ > t the following error bound in measuring t and t‟ holds:

  $$n\,(1-r)(t'-t) \le H(t') - H(t) \le (1+r)(t'-t)\,n$$

- ✓ At this condition, no jumps in hardware clocks allowed
- ✓ The most frequently used conditions are:
  - ❖ Monotonically increasing
  - ❖ Drift rate bounded between synchronization points
  - ❖ Clock may jump ahead at synchronization points

Working of Computer timer:

- To implement a clock in a computer a counter register and a holding register are used.
- The counter is decremented by a quartz crystals oscillator.
- When it reaches zero, an interrupted is generated and the counter is reloaded from the holding register.

Clock skew problem

To avoid the clock skew problem, two types of clocks are used:

- ❖ Logical clocks : to provide consistent event ordering
- ❖ Physical clocks : clocks whose values must not deviate from the real time by more than a certain amount.

Software based solutions for synchronising clocks

The following techniques are used to synchronize clocks:

- ✓ time stamps of real-time clocks
- ✓ message passing
- ✓ round-trip time (local measurement)

Based on the above mentioned techniques, the following algorithmsprovides clock synchronization:

- ↑ Cristian‟s algorithm
- ↑ Berkeley algorithm
- ↑ Network time protocol (Internet)

4.1.3 Cristian's algorithm

Cristian suggested the use of a time server, connected to a device that receives signals from a source of UTC, to synchronize computers externally. Round trip times between processes are often reasonably short in practice, yet theoretically unbounded. The practical estimation is possible if round-trip times are sufficiently short in comparison to required accuracy.

Principle:

- ∗ The UTC-synchronized time server S is used here.
- ∗ A process P sends requests to S and measures round-trip time T_{round} .
- ∗ In LAN, T_{round} should be around 1-10 ms .
- ∗ During this time, a clock with a 10-6 sec/sec drift rate varies by at most 10-8 sec.
- ∗ Hence the estimate of T_{round} is reasonably accurate .
- ∗ Now set clock to $t + ½ T_{round.}$

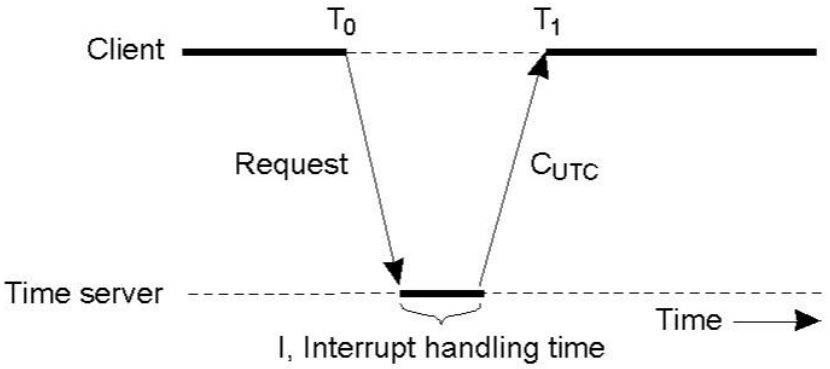


Fig 4.1 Cristian's algorithm

As per the given algorithm, the following results were found:

- ✓ Earliest time that S can have sent reply: t0 + min
- ✓ Latest time that S can have sent reply: t0 + T_{round} – min
- ✓ Total time range for answer: T_{round} - 2 * min
- ✓ Accuracy is ± (½T_{round} - min)

Problems in this system:

- Timer must never run backward.
- Variable delays in message passing / delivery occurs.

4.1.4 Berkeley algorithm

- Berkeley algorithm was developed to solve the problems of Cristian‟s algorithm.
- This algorithm does not need external synchronization.
- Master slave approach is used here.
- The master polls the slaves periodically about their clock readings.
- Estimate of local clock times is calculated using round trip.
- The average values are obtained from a group of processes.
- This method cancels out individual clock‟s tendencies to run fast and tells slave processes by which amount of time to adjust local clock
- In case of master failure, **master election algorithm** is used.

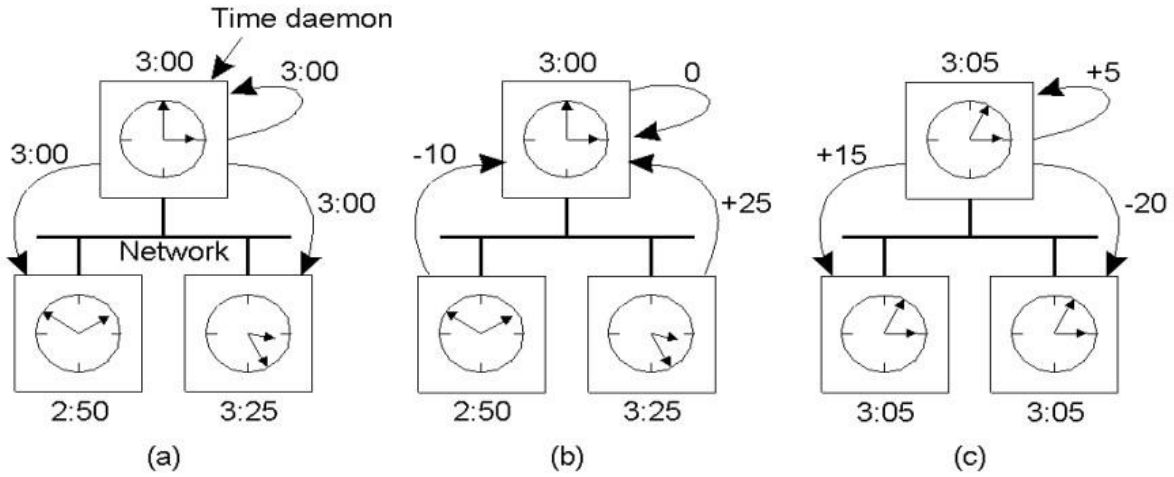


Fig 4.2: Berkeley Algorithm

In other words, the working of the algorithm can be summarized as,

- The time daemon asks all the other machines for their clock values.
- The machines answer the request.
- The time daemon tells everyone how to adjust their clock.

4.1.5 Network Time Protocol (NTP)

The Network Time Protocol defines architecture for a time service and a protocol to distribute time information over the Internet.

Features of NTP:

- ↑ To provide a service enabling clients across the Internet to be synchronized accurately to UTC.
- ↑ To provide a reliable service that can survive lengthy losses of connectivity.
- ↑ To enable clients to resynchronize sufficiently frequently to offset the rates of drift found in most computers.
- ↑ To provide protection against interference with the time service, whether malicious or accidental.

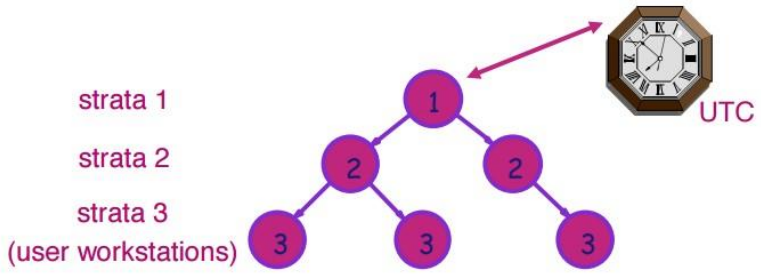


Fig 4.3: NTP strata

- The NTP service is provided b y a network of servers located across the Internet.
- Primary servers are connected directly to a time source such as a radio clock receiving UTC.
- Secondary servers are synchronized, with primary servers.
- The servers are connected in a logical hierarchy called a **synchronization subnet**.
- Arrows denote synchronization control, numbers denote strata. The levels are called **strata.**

Working:

- ↑ NTP follows a layered client-server architecture, based on UDP message passing.
- ↑ Synchronization is done at clients with higher strata number and is less accurate due to increased latency to strata 1 time server.
- ↑ If a strata 1 server fails, it may become a strata 2 server that is being synchronized though another strata 1 server.

Modes of NTP:

NTP works in the following modes:

- ❖ **Multicast:**
  - ✓ One computer periodically multicasts time info to all other computers on network.
  - ✓ These adjust clock assuming a very small transmission delay.
  - ✓ Only suitable for high speed LANs; yields low but usually acceptable synchronization.
- ❖ **Procedure-call:**
  - ✓ This is similar to Christian‟s protocol .

- ✓ Here the server accepts requests from clients.
- ✓ This is applicable where higher accuracy is needed, or where multicast is not supported by the network‟s hardware and software

❖ **Symmetric:**

- ✓ This is used where high accuracy is needed.

Working of Procedure call and symmetric modes:

- All messages carry timing history information.
- The history includes the local timestamps of send and receive of the previous NTP message and the local timestamp of send of this message
- For each pair i of messages (m, m‟) exchanged between two servers the following values are being computed

  - offseto_i : estimate for the actual offset between two clocks

  -delay d_i : true total transmission time for the pair of messages.

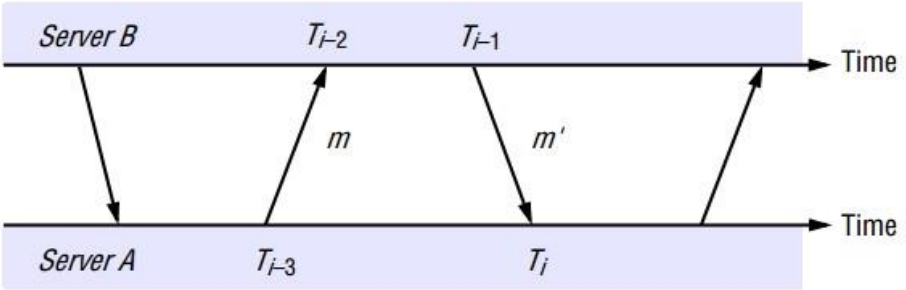


Fig 4.4: Message exchange between NTP

Delay and offset:

- ✓ Let o be the true offset of B‟s clock relative to A‟s clock, and let t and t‟ the true transmission times of m and m‟ (Ti , Ti-1 ... are not true time)
- ✓ The delay Ti-2 = Ti-3 + t + o…….. (1)

  Ti = Ti-1 + t‟ – o ……(2)

  which leads to di = t + t‟ = Ti-2 - Ti-3 + Ti - Ti-1 (clock errors zeroed out à (almost) true d)
- ✓ Offset oi = ½ (Ti-2 – Ti-3 + Ti-1 – Ti) (only an estimate)

Implementing NTP

- ❖ Statistical algorithms based on 8 most recent pairs are used in NTP to determine quality of estimates.
- ❖ The value of oi that corresponds to the minimum di is chosen as an estimate for o .
- ❖ Time server communicates with multiple peers, eliminates peers with unreliable data, favors peers with higher strata number (e.g., for primary synchronization partner selection).
- ❖ NTP phase lock loop model: modify local clock in accordance with observed drift rate.
- ❖ The experiments achieve synchronization accuracies of 10 msecs over Internet, and 1 msec on LAN using NTP

4.2 LOGICAL TIME AND LOGICAL CLOCKS

- ✓ Synchronization between two processes without any form of interaction is not needed.
- ✓ But when processes interact, they must be event ordered. This is done by logical clocks.
- ✓ Consider a distributed system with n processes, p_1, p_2, …p_n.
- ✓ Each process p_i executes on a separate processor without any memory sharing.
- ✓ Each p_i has a state s_i. The process execution is a sequence of events.
- ✓ As a result of the events, changes occur in the local state of the process.
- ✓ They either send or receive messages.

4.2.1 Lamport Ordering of Events

The partial ordering obtained by generalizing the relationship between two process is called as **happened-before relation or causal ordering or potential causal ordering**. This term was coined by Lamport. Happens-before defines a partial order of events in a distributed system. Some events can‟t be placed in the order.

- We say $e \rightarrow_i e'$ if e happens before e‟ at process i.
- $e \rightarrow e'$ is defined using the following rules:
  - Local ordering: $e \rightarrow e'$ if $e \rightarrow_i e'$ for any process i
  - Messages: send(m) → receive(m) for any message m
  - Transitivity: $e \rightarrow e''''$ if $e \rightarrow e'$ and $e' \rightarrow e''''$

4.2.2 Logical Clocks

A Lamport logical clock is a monotonically increasing software counter, whose value need bear no particular relationship to any physical clock.

- ✓ Lamport clock L orders events consistent with logical happens before ordering.

  If e → e‟, then L(e) < L(e‟)

- ✓ But the converse is not true.
- ✓

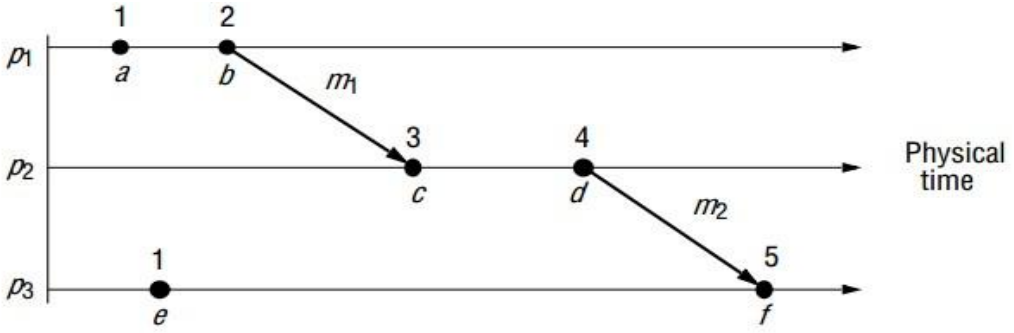


Fig 4.5: Lamport's timestamps for the three process

Lamport's Algorithm

- ∗ Assume that each process i keeps a local clock, L_i. There are three rules:
  1. At process i, increment L_i before each event.
  2. To send a message m at process i, apply rule 1 and then include the current local time in the message: i.e., send(m,L_i).
  3. To receive a message (m,t) at process j, set L_j = max(L_j,t) and then apply rule 1 before time-stamping the receive event.
- ∗ The global time L(e) of an event e is just its local time. For an event e at process i, L(e) = L_i(e).

Totally ordered logical clocks

- Many systems require a total-ordering of events, not a partial-ordering.
- Use Lamport‟s algorithm, but break ties using the process ID.
  - $L(e) = M * L_i(e) + i$

  Where M = maximum number of processes

Vector clocks

The main aim of vector clocks is to order in such a way to match causality. The expression, $V(e) < V(e‟)$ if and only if $e \rightarrow e‟$, holds for vector clocks, where V(e) is the vector clock for event e. For implementing vector clock, label each event by vector V(e) [c_1, c_2 …, c_n]. $c_i$is the events in process i that causally precede e.

- Each processor keeps a vector of values, instead of a single value.
- VC_i is the clock at process i; it has a component for each process in the system.

  $VC_i[i]$ corresponds to P_i„s local “time”.

  $VC_i[j]$ represents P_i„s knowledge of the “time” at P_j(the # of events that P_i knows have occurred at Pj
- Each processor knows its own “time” exactly, and updates the values of other processors‟ clocks based on timestamps received in messages.

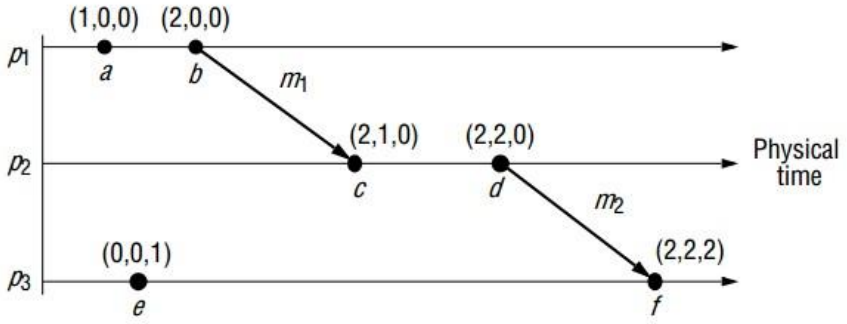


Fig 4.6: Vector Timestamps

Vector timestamps have the disadvantage, compared with Lamport timestamps, of taking up an amount of storage and message payload that is proportional to N, the number of processes.

4.3 GLOBAL STATES

This section checks whether a property is true in a distributed systems. This is checked for the following problems:

- ✓ distributed garbage collection
- ✓ deadlock detection
- ✓ termination detection
- ✓ Debugging

The global state of a distributed system consists of the local state of each process, together with the messages that are currently in transit, that is, that have been sent but not delivered.

Distributed Garbage Collection

- An object is considered to be garbage if there are no longer any references to it anywhere in the distributed system.

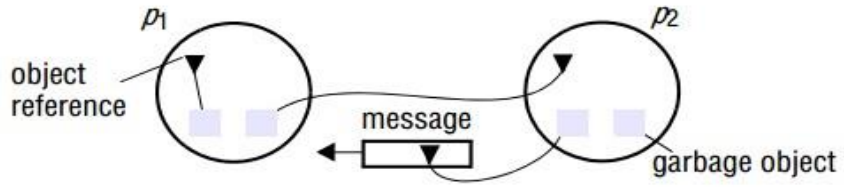


Fig 4. 7: Distributed Garbage Collection

- To check that an object is garbage, verify that there are no references to it anywhere.
- The process p1 has two objects that both have references :
  - ✓ One has a reference within p1 itself
  - ✓ p2 has a reference to the other.
- Process p2 has one garbage object, with no references to it anywhere in the system.
- It also has an object for which neither p1 nor p2 has a reference, but there is a reference to it in a message that is in transit between the processes.

Distributed deadlock detection:

- A distributed deadlock occurs when each of a collection of processes waits for another process to send it a message, and where there is a cycle in the graph of this „waits-for" relationship.
- In the following figure, processes p1 and p2 are each waiting for a message from the other, so this system will never make progress.

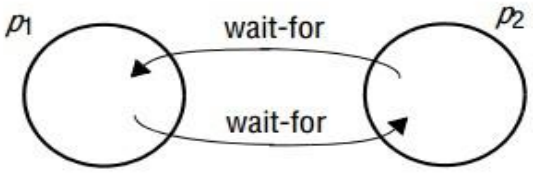


Fig 4.8: Distributed deadlock detection

Termination detection

- Detecting termination is a problem is to test whether each process has halted.
- Find whether a process is either active or passive.
- A passive process is not engaged in any activity of its own but is prepared to respond with a value requested by the other.
- The phenomena of termination and deadlock are similar:
  - ↑ A deadlock may affect only a subset of the processes in a system, whereas all processes must have terminated
  - ↑ The pro n a deadlock cycle: a deadloc ther action, for which another in any activity.

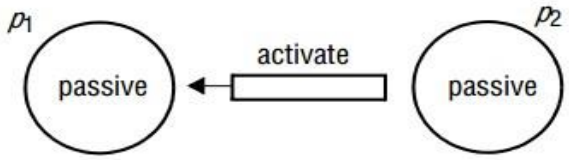


Fig 4.9: Termination of process

Distributed debugging:

Consider an application with process pi and a variable x, where i=1, 2, …, N. As the program executes the variables may change the value. The value must be within the range. The relationships between variables must be calculated only for the variables which executes at same time.

4.3.1 Global States and Consistency cuts (Distributed Snapshots)

Distributed Snapshot represents a state in which the distributed system might have been in. A snapshot of the system is a single configuration of the system.

A distributed snapshot should reflect a consistent state. A global state is consistent if it could have been observed by an external observer. For a successful Global State, all states must be consistent

- If we have recorded that a process P has received a message from a process Q, then we should have also recorded that process Q had actually send that message.
- Otherwise, a snapshot will contain the recording of messages that have been received but never sent.
- The reverse condition (Q has sent a message that P has not received) is allowed.

The notion of a global state can be graphically represented by what is called a **cut**. A cut represents the last event that has been recorded for each process.

The history of each process if given by:

$$\text{history}(p_i) = h_i = < e_i^0, e_i^1, e_i^2, ... >$$

Each event either is an internal action of the process. We denote by s_i^kt he state of process p_i immediately before the kth event occurs. The state $s_i$in the global state S correspondin g to the cut C is that of p_i immediately after the last event processed by p_i in the cut – ei^{ci}. The set of events ei^{ci} is called the frontier of the cut.

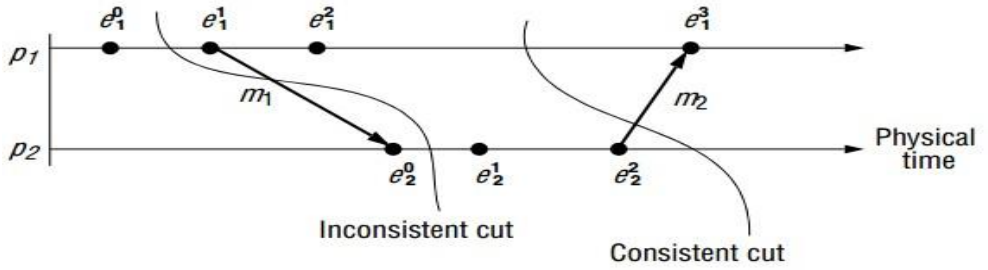


Fig 4.10: Types of cuts

In the above figure, the $<e_1^1, e_2^0>$and $<e_2^2, e_1^3>$ are the frontiers.The $<e_1^1, e_2^0>$ is inconsistent because at p2 it includes the receipt of the message m1 , but at p1 it does not include the sending of that message. The frontier $<e_2^2, e_1^3>$ is consistent since it includes both the sending and the receipt of message m1 and the sending but not the receipt of message m2.

A **consistent global state** is one that corresponds to a consistent cut. We may characterize the e execution of a distributed s yste m as a series of transitions between global states of the system:

$$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow \ldots$$

A **run** is a total ordering of all the events in a global history that is consistent with each local history"s ordering. A **linearization** or consistent run is an ordering of the events in a global history that is consistent with this happened-before relation o on H. A linearization is also a run.

4.3.2 Global state predicates, stability, safety and liveness

- Detecting a condition such as deadlock or termination amounts to evaluating a global state predicate.
- A **global state predicate** is a function that maps from the set of global states of processes in the system
- One of the useful characteristics of the predicates associated with the state of an object being garbage, of the system being deadlocked or the system being terminated is that they are all stable: once the system enters a state in which the predicate is True, it remains True in all future states reachable from that state.
- By contrast, when we monitor or debug an application we are often interested in non-stable predicates, such as that in our example of variables whose difference is supposed to be bounded.

4.3.3 Snapshot algorithm of Chandy and Lamport

Assumptions of the algorithm

- The algorithm to determine global states records process states locally and the states are collected by a designated server.
- No failures in c
- Unidirectional
- There is always
- Global snapsho
- No process acti

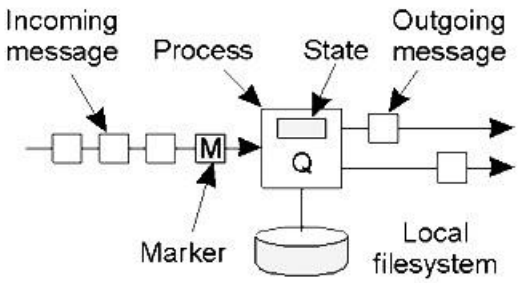


Fig 4.11: Process and organization in distributed systems

Algorithm:

- If a process Q receives the marker requesting a snapshot for the first time, it considers the process that sent the marker as its predecessor.

- When Q completes its part of the snapshot, it sends its predecessor a DONE message.
- By recursion, when the initiator of the distributed snapshot has received a DONE message from all its successors, it knows that the snapshot has been completely taken.

4.4 COORDINATION AND AGREEMENT

When more that one process run in an execution environment, they need to coordinate their actions. To solve problems in coordination avoid fixed master-salve relationship to avoid single points of failure for fixed master. Distributed mutual exclusion for resource sharing can also be used.

When a collection of process share resources, mutual exclusion is needed to prevent interference and ensure consistency. In this case, there is no need for shared variables or facilities are provided by single local kernel to solve it. This solution is based on message passing. The main issue to be addressed while implementing the above method is the failure.

4.4.1 Failure Assumptions and failure Detectors

- ✓ The message passing paradigm is based on reliable communication channels.
- ✓ But there can be process failures (i.e.) the whole process crashes.
- ✓ The failure can be detected when the object/code in a process that detects failures of other processes.
- ✓ There are two types of failure detectors: unreliable failure detector and reliable failure detector.
- ✓ The following are features of the unreliable failure detectors:
  - unsuspected or suspected (i.e.) there can be no evidence of failure
  - each process sends ``alive'' message to everyone else
  - not receiving ``alive'' message after timeout
  - This is present in most practical systems
- ✓ The following are features of the reliable failure detectors:
  - unsuspected or failure
  - They are present in synchronous system

4.5 DISTRIBUTED MUTUAL EXCLUSION

The mutual exclusion makes sure that concurrent process access shared resources or data in a serialized way. If a process, say Pi , is executing in its critical section, then no other processes can be executing in their critical sections.

Distributed mutual exclusion provide critical region in a distributed environment.

4.5.1 Algorithms for Distributed Mutual Exclusion

Consider there are N processes and the processes do not fail. The basic assumption is the message delivery system is reliable. So the methods for the critical region are:

- ✓ **enter() :** Enter the critical section block if necessary
- ✓ **resourceAccesses():**Access the shared resources
- ✓ **exit():** Leaves the critical section. Now other processes can enter critical section.

The following are the requirements for Mutual Exclusion (ME):

- ➢ **[ME1] safety**: only one process at a time
- ➢ **[ME2] liveness**: eventually enter or exit
- ➢ **[ME3] happened-before ordering:** ordering of enter() is the same as HB ordering

The second requirement implies freedom from both deadlock and starvation. Starvation involves fairness condition.

Performance Evaluation:

The following are the criteria for performance measures:

- ❖ **Bandwidth consumption**, which is proportional to the number of messages sent in each entry and exit operations.
- ❖ The **client delay** incurred by a process at each entry and exit operation.
- ❖ **Throughput of the system**: Rate at which the collection of processes as a whole can access the critical section.

4.5.2 Central Sever Algorithm

- ✓ This employs the simplest way to grant permission to enter the critical section by using a server.
- ✓ A process sends a request message to server and awaits a reply from it.

- ✓ If a reply constitutes a token signifying the permission to enter the critical section.
- ✓ If no other process has the token at the time of the request, then the server replied immediately with the token.
- ✓ If token is currently held by another process, then the server does not reply but queues the request.
- ✓ Client on exiting the critical section, a message is sent to server, giving it back the token.

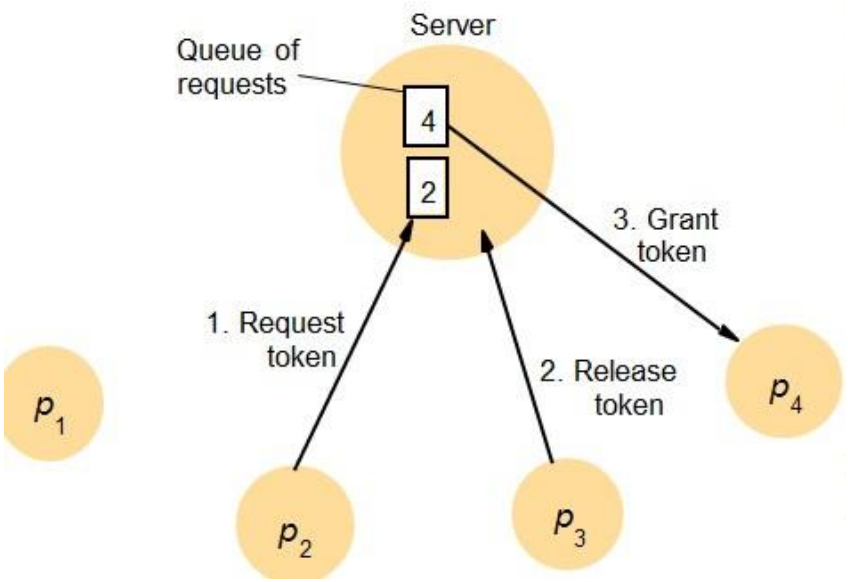


Fig 4. 12: Central Server Algorithm

The central server algorithm fulfils ME1 and ME2 but not ME3 (i.e.) safety and liveness is ensured but ordering is not satisfied. Also the performance of the algorithm is measured as follows:

- ➢ Bandwidth: This is measured by entering and exiting messages. Entering takes two messages (request followed by a grant) which are delayed by the round-trip time. Exiting takes one release message, and does not delay the exiting process.
- ➢ Throughput is measured by synchronization delay, round-trip of a release message and grant message.

4.5.3 Ring Based Algorithm

- ❖ This provides a simplest way to arrange mutual exclusion between N processes without requiring an additional process is arrange them in a logical ring.

- ❖ Each process pi has a communication channel to the next process in the ring as follows: p(i+1)/mod N.
- ❖ The unique token is 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 CS when it receives the token, then it immediately forwards the token to its neighbor.
- ❖ A process requires the token waits until it receives it, but retains it.
- ❖ To exit the critical section, the process sends the token on to its neighbor.

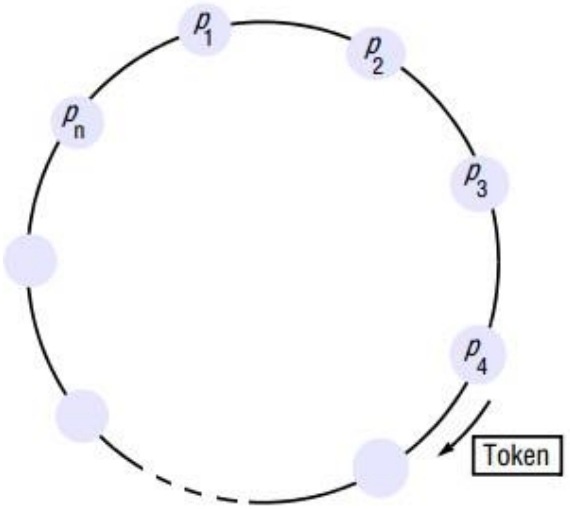


Fig 4.13: Ring based algorithm

This algorithm satisfies ME1 and ME2 but not ME (i.e.) safety and liveness are satisfied but not ordering. The performance measures include:

- ➢ **Bandwidth:** continuously consumes the bandwidth except when a process is inside the CS. Exit only requires one message.
- ➢ **Delay:** experienced by process is zero message(just received token) to N messages(just pass the token).
- ➢ **Throughput:** synchronization delay between one exit and next entry is anywhere from 1(next one) to N (self) message transmission.

4.5.4 Multicast Synchronisation

- This exploits mutual exclusion between N peer processes based upon multicast.
- 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 condition under which a process replies to a request are designed to ensure ME1 ME2 and ME3 are met.
- Each process pi keeps a Lamport clock. Message requesting entry are of the form<T, pi>.
- Each process records its state of either RELEASE, WANTED or HELD in a variable state.
  - ↑ If a process requests entry and all other processes is RELEASED, then all processes reply immediately.
  - ↑ If some process is in state HELD, then that process will not reply until it is finished.
  - ↑ If some process is in state WANTED and has a smaller timestamp than the incoming request, it will queue the request until it is finished.
- If two or more processes request entry at the same time, then whichever bears the lowest timestamp will be the first to collect N-1 replies.

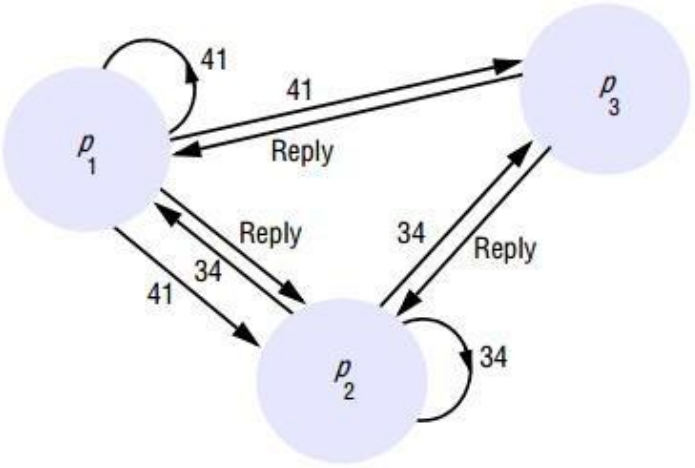


Fig 4.13: Multicast Synchronisation

- In the above figure, P1 and P2 request CS concurrently.
- The timestamp of P1 is 41 and for P2 is 34.
- When P3 receives their requests, it replies immediately.
- When P2 receives P1"s request, it finds its own request has the lower timestamp, and so does not reply, holding P1 request in queue.
- However, P1 will reply. P2 will enter CS. After P2 finishes, P2 reply P1 and P1 will enter CS.
- Granting entry takes 2(N-1) messages, N-1 to multicast request and N-1 replies.

Performance Evaluation:

- ❖ Bandwidth consumption is high.
- ❖ Client delay is again 1 round trip time
- ❖ Synchronization delay is one message transmission time.

4.5.5 Maekawa's Voting Algorithm

- In this algorithm, it is not necessary for all of its peers to grant access. Only need to obtain permission to enter from subsets of their peers, as long as the subsets used by any two processes overlap.
- Think of processes as voting for one another to enter the CS. A candidate process must collect sufficient votes to enter.
- Processes in the intersection of two sets of voters ensure the safety property ME1 by casting their votes for only one candidate.
- A voting set Vi associated with each process pi.
- There is at least one common member of any two voting sets, the size of all voting set are the same size to be fair.
- The optimal solution to minimizes K is K~sqrt(N) and M=K.
- The algorithm is summarized as follows:

  $V_i \subseteq \{P_1, P_2, \ldots P_N\}$

  Such that for all i, j = 1, 2, … N

  $Pi \in V_i$

  $V_i \cap Vj \neq \phi$

  $|V_i| = K$

  Each process is contained in M of the voting set V_i

Maekawa's Voting Algorithm

On initialization

state := RELEASED;

voted := FALSE;

For p_i to enter the critical section

state := WANTED;

```
        Multicast request to all processes in Vi;
        Wait until (number of replies received = K);
        state := HELD;
On receiptof a request frompi at pj
        if (state = HELD orvoted = TRUE)
        then
            queue request from pi without replying;
        else
            send reply to pi;
            voted := TRUE;
        end if
For pi to exit the critical section
        state := RELEASED;
        Multicast release to all processes in Vi;
On receiptof a release frompi at pj
        if (queue of requests is non-empty)
        then
            remove head of queue – from pk, say;
            send reply to pk;
            voted := TRUE;
        else
            voted := FALSE;
        end if
```

- The ME1 is met.
- If two processes can enter CS at the same time, the processes in the intersection of two voting sets would have to vote for both.
- The algorithm will only allow a process to make at most one vote between successive receipts of a release message.
- This algorithm is deadlock prone.

- If three processes concurrently request entry to the CS, then it is possible for p1 to reply to itself and hold off p2; for p2 rely to itself and hold off p3; for p3 to reply to itself and hold off p1.
- Each process has received one out of two replies, and none can proceed.
- If process queues outstanding request in happen-before order, ME3 can be satisfied and will be deadlock free.

Performance Evaluation:

- Bandwidth utilization is 2sqrt(N) messages per entry to CS and sqrt(N) per exit.
- Client delay is the same as Ricart and Agrawala‟salgorithm, one round-trip time.
- Synchronization delay is one round-trip time which is worse than R&A

4.5.6 Fault Tolerance

- ❖ The reactions of the algorithms when messages are lost or when a process crashes is fault tolerance.
- ❖ None of the algorithm that we have described would tolerate the loss of messages if the channels were unreliable.
- ❖ The ring-based algorithm cannot tolerate any single process crash failure.
- ❖ Maekawa‟salgorithm can tolerate some process crash failures: if a crashed process is not in a voting set that is required.
- ❖ The central server algorithm can tolerate the crash failure of a client process that neither holds nor has requested the token.

4.6 ELECTIONS

An algorithm for choosing a unique process to play a particular role is called an election algorithm.

The requirements for election algorithms:

- **E1(safety):** a participant pi haselected$_i$ = ⊥ or elected P where ⊥ indicates a value that is not defined.
- **E2(liveness):** All processes Pi participate in election process and eventually setelected$_i$!=⊥ or crash.

4.6.1 Ring based Election Algorithm

- All the processes arranged in a logical ring.
- Each process has a communication channel to the next process.
- All messages are sent clockwise around the ring.
- Assume that no failures occur, and system is asynchronous.
- The ultimate goal is to elect a single process coordinator which has the largest identifier.

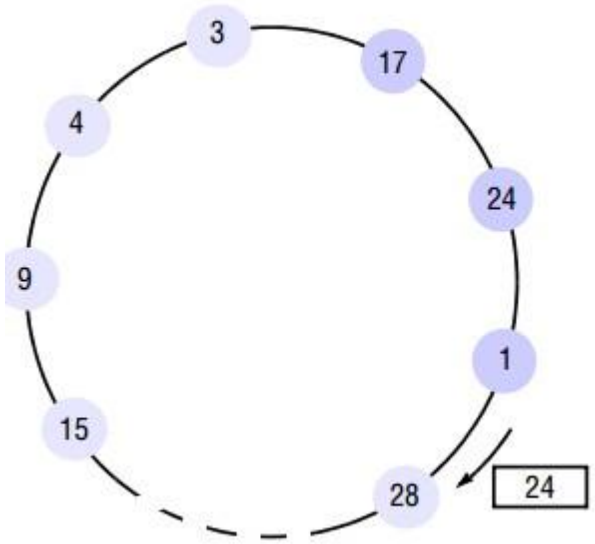


Fig 4.14: Election process using Ring based election algorithm

Steps in election process:

1. Initially, every process is marked as non-participant. Any process can begin an election.
2. The starting processes marks itself as participant and place its identifier in a message to its neighbour.
3. A process receives a message and compares it with its own. If the arrived identifier is larger, it passes on the message.
4. If arrived identifier is smaller and receiver is not a participant, substitute its own identifier in the message and forward if. It does not forward the message if it is already a participant.
5. On forwarding of any case, the process marks itself as a participant.

6. If the received identifier is that of the receiver itself, then this process‟ s identifier must be the greatest, and it becomes the coordinator.

7. The coordinator marks itself as non-participant set elected_i and sends an elected message to its neighbour enclosing its ID.

8. When a process receives elected message, marks itself as a non-participant, sets its variable elected_i and forwards the message.

The election was started by process 17. The highest process identifier encountered so far is 24.

Requirements:

- ↑ E1 is met. All identifiers are compared, since a process must receive its own ID back before sending an elected message.
- ↑ E2 is also met due to the guaranteed traversals of the ring.
- ↑ Tolerates no failure makes ring algorithm of limited practical use.

Performance Evaluation

- If only a single process starts an election, the worst-performance case is then the anti-clockwise neighbour has the highest identifier. A total of N-1 messages is used to reach this neighbour. Then further N messages are required to announce its election. The elected message is sent N times. Making 3N-1 messages in all.
- Turnaround time is also 3N-1 sequential message transmission time.

4.6.2 Bully Algorithm

- ❖ This algorithm allows process to crash during an election, although it assumes the message delivery between processes is reliable.
- ❖ Assume that the system is synchronous to use timeouts to detect a process failure and each process knows which processes have higher identifiers and that it can communicate with all such processes.
- ❖ In this algorithm, there are three types of messages:
  - **Election message:** This is sent to announce an election message. A process begins an election when it notices, through timeouts, that the coordinator has failed. T=2Ttrans+Tprocess From the time of sending
  - **Answermessage:** This is sent in response to an election message.
  - **Coordinatormessage**: This is sent to announce the identity of the elected process.

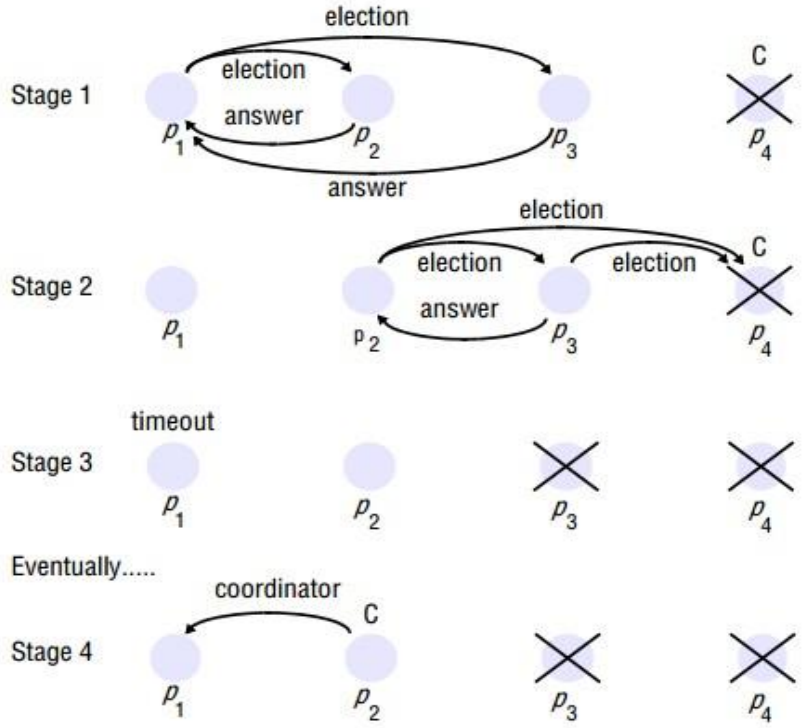


Fig 4.15: Stages in Bully Algorithm

Election process:

- The process begins a election by sending an election message to these processes that have a higher ID and awaits an answer in response.
- If none arrives within time T, the process considers itself the coordinator and sends coordinator message to all processes with lower identifiers.
- Otherwise, it waits a further time T‟ for coordinator message to arrive. If none, begins another election.
- If a process receives a coordinator message, it sets its variable elected_i to be the coordinator ID.
- If a process receives an election message, it send back an answer message and begins another election unless it has begun one already.

Requirements:

- E1 may be broken if timeout is not accurate or replacement.
- Suppose P3 crashes and replaced by another process. P2 set P3 as coordinator and P1 set P2 as coordinator.
- E2 is clearly met by the assumption of reliable transmission.

Performance Evaluation

- Best case the process with the second highest ID notices the coordinator‟s failure. Then it can immediately elect itself and send N-2 coordinator messages.
- The bully algorithm requires O(N^2) messages in the worst case - that is, when the process with the least ID first detects the coordinator‟s failure. For then N-1 processes altogether begin election, each sending messages to processes with higher ID.

4.7 TRANSACTION AND CONCURRENCY CONTROL

Fundamentals of transactions and concurrency control

- ✓ A transaction is generally atomic.
- ✓ The state of the transaction being done is not visible. If it is not done completely, any changes it made will be undone. This is known as rollback.
- ✓ Concurrency is needed when multiple users want to access the same data at the same time.
- ✓ Concurrency control (CC) ensures that correct results for parallel operations are generated.
- ✓ CC provides rules, methods, design methodologies and theories to maintain the consistency of components operating simultaneously while interacting with the same object.

A Transaction defines a sequence of server operations that is guaranteed to be atomic in the presence of multiple clients and server crash.

All concurrency control protocols are based on serial equivalence and are derived from rules of conflicting operations:

- ➢ Locks used to order transactions that access the same object according to request order.
- ➢ Optimistic concurrency control allows transactions to proceed until they are ready to commit, whereupon a check is made to see any conflicting operation on objects.
- ➢ Timestamp ordering uses timestamps to order transactions that access the same object according to their starting time.

Synchronisation without transactions

- ↑ Synchronization without transactions are done with multiple threads.
- ↑ The use of multiple threads is beneficial to the performance. Multiple threads may access the same objects.

- In Java, Synchronized keyword can be applied to method so only one thread at a time can access an object.
- If one thread invokes a synchronized method on an object, then that object is locked, another thread that invokes one of the synchronized method will be blocked.

Enhancing Client Cooperation by Signaling

- The clients may use a server as a means of sharing some resources.
- In some applications, threads need to communicate and coordinate their actions by signaling.

Failure modes in transactions

- The following are the failure modes: Writes to permanent storage may fail, either by writing nothing or by writing wrong values.
- Servers may crash occasionally.
- There may be an arbitrary delay before a message arrives.

4.8 TRANSACTIONS

- Transaction originally from database management systems.
- Clients require a sequence of separate requests to a server to be atomic in the sense that:
  - They are free from interference by operations being performed on behalf of other concurrent clients.
  - Either all of the operations must be completed successfully or they must have no effect at all in the presence of server crashes**.**

Atomicity

- All or nothing: a transaction either completes successfully, and effects of all of its operations are recorded in the object, or it has no effect at all.
  - **Failure atomicity**: effects are atomic even when server crashes
  - **Durability:** after a transaction has completed successfully, all its effects are saved in permanent storage for recover later.

Isolation

Each transaction must be performed without interference from other transactions. The intermediate effects of a transaction must not be visible to other transactions.

4.8.1 Concurrency Control

Serial Equivalence

- ✓ If these transactions are done one at a time in some order, then the final result will be correct.
- ✓ If we do not want to sacrifice the concurrency, an interleaving of the operations of transactions may lead to the same effect as if the transactions had been performed one at a time in some order.
- ✓ We say it is a serially equivalent interleaving.
- ✓ The use of serial equivalence is a criterion for correct concurrent execution to prevent lost updates and inconsistent retrievals.

Conflicting Operations

- ✓ When we say a pair of operations conflicts we mean that their combined effect depends on the order in which they are executed. E.g. read and write
- ✓ There are three ways to ensure serializability:
  - ➢ Locking
  - ➢ Timestamp ordering
  - ➢ Optimistic concurrency control

Process 1	**Process 2**	**Conflict**	**Reason**
Read	Read	No	-
Read	Write	Yes	The result of these operations dependon their order of execution.
Write	Write	Yes	The result of these operations depend on their order of execution.

4.9 NESTED TRANSACTIONS

- ❖ Nested transactions extend the transaction model by allowing transactions to be composed of other transactions.
- ❖ Thus several transactions may be started from within a transaction, allowing transactions to be regarded as modules that can be composed as required.
- ❖ The outermost transaction in a set of nested transactions is called the **top-level transaction.**
- ❖ Transactions other than the top-level transaction are called **sub-transactions.**

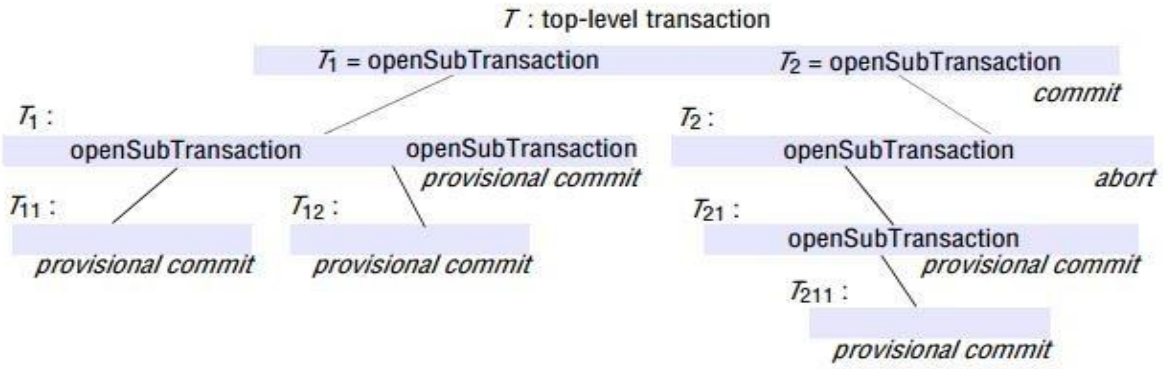


Fig 4.16: Nested Transactions

- T is a top-level transaction that starts a pair of sub-transactions, T1 and T2.
- The sub-transaction T1 starts its own pair of sub-transactions, T11 and T22.
- Also, sub-transaction T2 starts its own sub-transaction, T21, which starts another sub-transaction, T211.
- A sub-transaction appears to be atomic to its parent with respect to transaction failures
  and to concurrent access.
- Sub-transactions at the same level can run concurrently, but their access to common objects is serialized.
- Each sub-transaction can fail independently of its parent and of the other sub-transactions.
- When a sub-transaction aborts, the parent transaction can choose an alternative sub-transaction to complete its task.
- If one or more of the sub-transactions fails, the parent transaction could record the fact and then commit, with the result that all the successful child transactions commit.
- It could then start another transaction to attempt to redeliver the messages that were not sent the first time.

Advantages of Nested Transactions

- Sub- transactions at same level can run concurrently.
- Sub- transactions can commit or abort independently.

The rules for committing of nested transactions are:

- A transaction may commit or abort only after its child transactions have completed.
- When a sub-transaction completes, it makes an independent decision either to commit provisionally or to abort. Its decision to abort is final.
- When a parent aborts, all of its sub-transactions are aborted.
- When a sub -transaction aborts, the parent can decide whether to abort or not.
- If the top-level transaction commits, then all of the sub-transactions that have provisionally committed can commit too, provided that none of their ancestors has aborted.

4.10 LOCKS

- A simple example of a serializing mechanism is the use of exclusive locks.
- Server can lock any object that is about to be used by a client.
- If another client wants to access the same object, it has to wait until the object is unlocked in the end.
- There are two types of locks: Read and Write.
- Read locks are shared locks (i.e.) they do not bring about conflict.
- Write locks are exclusive locks, since the order in which write is done may give rise to a conflict.

Lock	**Read conflict**	**Write conflict**
None	No	No
Read	No	Yes
Write	Yes	Yes

- An object can be read and write.
- From the compatibility table, we know pairs of read operations from different transactions do not conflict.
- So a simple exclusive lock used for both read and write reduces concurrency more than necessary. (Many readers/Single writer).

The following rules could be framed:

1. If T has already performed a read operation, then a concurrent transaction U must not write until T commits or aborts.
2. If T already performed a write operation, then concurrent U must not read or write until T commits or aborts.

Lock implementation

- The granting of locks will be implemented by a separate object in the server called the lock manager.
- The lock manager holds a set of locks, for example in a hash table.
- Each lock is an instance of the class Lock and is associated with a particular object.
- Each instance of Lock maintains the following information in its instance variables:
  - the identifier of the locked object
  - the transaction identifiers of the transactions that currently hold the lock
  - a lock type

Two phase Locking

The basic two-phase locking (2PL) protocol states:

- ✓ A transaction T must hold a lock on an item x in the appropriate mode before T accesses x.
- ✓ If a conflicting lock on x is being held by another transaction, T waits.
- ✓ Once T releases a lock, it cannot obtain any other lock subsequently.

In two phase locking, a transaction is divided into two phases:

- ✓ A growing phase (obtaining locks)
- ✓ A shrinking phase (releasing locks)

This lock ensures conflict serializability.**Lock-point** is the point where the transaction obtains all the locks.With 2PL, a schedule is conflict equivalent to aa serial schedule ordered by the lock-point of the transactions.

Strict Two Phase Locking

- ➢ Because transaction may abort, strict execution are needed to prevent dirty reads and premature writes, which are caused by read or write to same object accessed by another earlier unsuccessful transaction that already performed an write operation.

- So to prevent this problem, a transaction that needs to read or write an object must be delayed until other transactions that wrote the same object have committed or aborted.
- The rule in Strict Two Phase Locking:
  - ↑ Any locks applied during the progress of a transaction are helduntil the transaction commits or aborts.

$$D = \begin{bmatrix} T1 & T2 \\ S(A) & \\ R(A) & \\ & S(A) \\ & R(A) \\ & X(B) \\ & R(B) \\ & W(B) \\ & Commit \\ X(C) & \\ R(C) & \\ W(C) & \\ Commit & \end{bmatrix}$$

Fig 4. 16: Strict two phase locking between T1 and T2

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.
   (d) If the object has already been locked in the same transaction, the lock will be promoted if necessary and the operation proceeds. (Where promotion is prevented by a conflicting lock, rule (b) is used.)
2. When a transaction is committed or aborted, the server unlocks all objects it locked for the transaction

A transaction with a read lock that is shared by other transactions cannot promote its read lock to a write lock, because write lock will conflict with other read locks.

4.10.1 Deadlocks

Deadlock is a state in which each member of a group of transactions is waiting for some other member to release a lock.

- A wait-for graph can be used to represent the waiting relationships between current transactions.
- In a wait-for graph the nodes represent transactions and the edges represent wait-for relationships between transactions – there is an edge from node T to node U when transaction T is waiting for transaction U to release a lock.

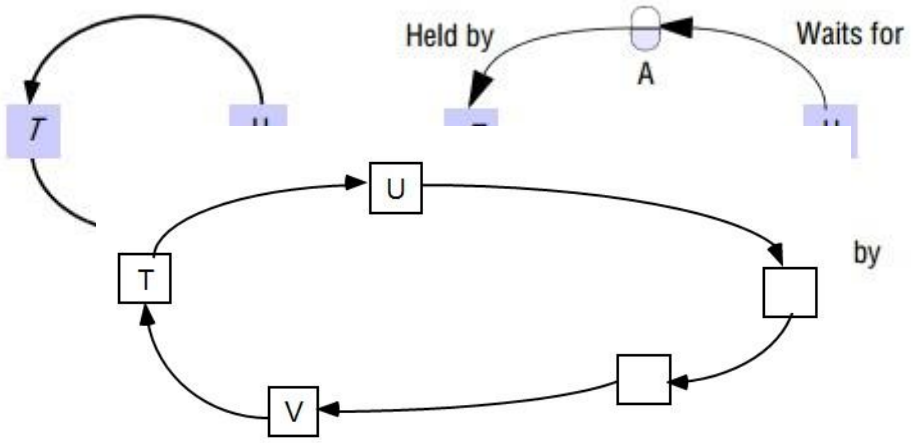


Fig 4.18: A cycle in wait for graph

Deadlock prevention:

- Simple way is to lock all of the objects used by a transaction when it starts.
- It should be done as an atomic action to prevent deadlock. a. inefficient, say lock an object you only need for short period of time. b.
- Hard to predict what objects a transaction will require.
- Judge if system can remain in a Safe state by satisfying a certain resource request. Banker‟s algorithm.
- Order the objects in certain order.
- Acquiring the locks need to follow this certain order.

Safe State

- ❖ System is in safe state if there exists a sequence <P_1, P_2, …, P_n> of ALL the processes is the systems such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with j <i.
- ❖ If a system is in safe state, then there are no deadlocks.
- ❖ If a system is in unsafe state, then there is possibility of deadlock.
- ❖ Avoidance is to ensure that a system will never enter an unsafe state.

Deadlock Detection

- Deadlock 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.
  - o If lock manager blocks a request, an edge can be added. Cycle should be checked each time a new edge is added.
  - o One transaction will be selected to abort in case of cycle. Age of transaction and number of cycles involved when selecting a victim
- Timeouts is commonly used to resolve deadlock. Each lock is given a limited period in which it is invulnerable. After this time, a lock becomes vulnerable.
  - – If no other transaction is competing for the object, vulnerable object remained locked. However, if another transaction is waiting, the lock is broken.

Disadvantages:

- Transaction aborted simply due to timeout and waiting transaction even if there is no deadlock.
- Hard to set the timeout time

4.11 OPTIMISTIC COCURRENCY CONTROL

The locking and serialization of transaction has numerous disadvantages.:

- ✓ Lock maintenance represents an overhead that is not present in systems that do not support concurrent access to shared data. Locking sometimes are only needed for some cases with low probabilities.
- ✓ The use of lock can result in deadlock. Deadlock prevention reduces concurrency severely. The use of timeout and deadlock detection is not ideal for interactive programs.
- ✓ To avoid cascading aborts, locks cannot be released until the end of the transaction. This may reduce the potential for concurrency.

- It is based on observation that, in most applications, the likelihood of two clients" transactions accessing the same object is low.
- Transactions are allowed to proceed as though there were no possibility of conflict with other transactions until the client completes its task and issues a closeTransaction request.
- When conflict arises, some transaction is generally aborted and will need to be restarted by the client.
- Each transaction has the following phases:
  - **Working phase:**
    - Each transaction has a tentative version of each of the objects that it updates.
    - This is a copy of the most recently committed version of the object.
    - The tentative version allows the transaction to abort with no effect on the object, either during the working phase or if it fails validation due to other conflicting transaction.
    - Several different tentative values of the same object may coexist.
    - In addition, two records are kept of the objects accessed within a transaction, a read set and a write set containing all objects either read or written by this transaction.
    - Read are performed on committed version (no dirty read can occur) and write record the new values of the object as tentative values which are invisible to other transactions.
  - **Validation phase:**
    - When closeTransaction request is received, the transaction is validated to establish whether or not its operations on objects conflict with operations of other transaction on the same objects.
    - If successful, then the transaction can commit.
    - If fails, then either the current transaction or those with which it conflicts will need to be aborted.
  - **Update phase:**
    - If a transaction is validated, all of the changes recorded in its tentative versions are made permanent.
    - Read-only transaction can commit immediately after passing validation.
    - Write transactions are ready to commit once the tentative versions have been recorded in permanent storage.

Validation of Transactions

- 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.
- Each transaction is assigned a number when it enters the validation phase (when the client issues a closeTransaction).
- Such number defines its position in time.
- A transaction always finishes its working phase after all transactions with lower numbers.
- That is, a transaction with the number Ti always precedes a transaction with number Tj if i < j.
- The validation test on transaction Tv is based on conflicts between operations in pairs of transaction Ti and Tv, for a transaction Tv to be serializable with respect to an overlapping transaction Ti, their operations must conform to the below rules.

Rule No	**T_v**	**T_1**	**Rule**
1	Write	Read	T_i must not read objects written by T_v
2	Read	Write	Tvmust not read objects written by T_i
3	Write	Write	T_i must not read objects written by $T_v$and $T_v$must not read objects written by Ti.

Types of Validation

- **Backward Validation:** checks the transaction undergoing validation with other preceding overlapping transactions- those that entered the validation phase before it.
- **Forward Validation:** checks the transaction undergoing validation with other later transactions, which are still active

Backward Validation	**Forward Validation**

Backward validation of transaction T_v
boolean valid = true;
for (intT_i = startTn+1; T_i<= finishTn; T_i++){
if (read set of T_v intersects write set of T_i) valid = false;
}
	Forward validation of transaction T_v
	boolean valid = true;
	for (intT_{id} = active1; T_{id}<= activeN; T_{id}++){
	if (write set of T_v intersects read set of T_{id}) valid = false;
	}

Starvation

- When a transaction is aborted, it will normally be restarted by the client program.
- There is no guarantee that a particular transaction will ever pass the validation checks, for it may come into conflict with other transactions for the use of objects each time it is restarted.

  The prevention of a transaction ever being able to commit is called starvation.

4.12 TIMESTAMP ORDERING

- Timestamps may be assigned from the server‟s clock or a counter that is incremented whenever a timestamp value is issued.
- Each object has a write timestamp and a set of tentative versions, each of which has a write timestamp associated with it; and a set of read timestamps.
- The write timestamps of the committed object is earlier than that of any of its tentative versions, and the set of read timestamps can be represented by its maximum member.
- Whenever a transaction‟s write operation on an object is accepted, the server creates a new tentative version of the object with write timestamp set to the transaction timestamp. Whenever a read operation is accepted, the timestamp of the transaction is added to its set of read timestamps.
- When a transaction is committed, the values of the tentative version become the values of the object, and the timestamps of the tentative version become the write timestamp of the corresponding object.
- Each request from a transaction is checked to see whether it conforms to the operation conflict rules.
- Conflict may occur when previous done operation from other transaction Ti is later than current transaction Tc. It means the request is submitted too late.

Rule	**T_c**	**T_i**	**Description**
1	Write	Read	T_c must not write an object that has been read by any T_i where $T_i > T_c$this requires that $T_c >=$ the maximum read timestamp of the object.
2	Write	Write	T_c must not write an object that has been written by any T_i where $T_i > T_c$this requires that $T_c >$ the write timestamp of the committed object.

3	Read	Write	T_c must not read an object that has been written by any T_i where $T_i > T_c$this requires that T_c>the write timestamp of the committed object.

Timestamp ordering by read rule:

```
if ( Tc> write timestamp on committed version of D) {
    let Dselected be the version of D with the maximum write timestamp ≤ Tc
    if (Dselected is committed)
            perform read operation on the version Dselected
    else
            Wait until the transaction that made version Dselected commits or aborts
            then reapply the read rule
} else
    Abort transaction Tc
```

Timestamp ordering by write rule

```
if (Tc ≥ maximum read timestamp on D&&
    Tc> write timestamp on committed version of D)
            perform write operation on tentative version of D with write timestamp Tc
else /* write is too late */
    Abort transaction Tc
```

Multi-version timestamp ordering

- ✓ In multi-version timestamp ordering, a list of old committed versions as well as tentative versions is kept for each object.
- ✓ This list represents the history of the values of the object.
- ✓ The benefit of using g multiple versions is that read operations that arrive too late need not be rejected.
- ✓ Each version has a read timestamp recording the largest timestamp of any transaction that has read from it in addition to a write timestamp.
- ✓ As before, whenever a write operation is accepted, it is directed to a tentative version with the write timestamp of the transaction.

- ✓ Whenever a read operation is carried out, it is directed to the version with the largest write timestamp less than the transaction timestamp.
- ✓ If the transaction timestamp is larger than the read timestamp of the version being used, the read timestamp of the version is set to the transaction timestamp.
- ✓ When a read arrives late, it can be allowed to read from an old committed version, so there is no need to abort late read operations.
- ✓ In multi-version timestamp ordering, read operations are always permitted, although they may have to wait for earlier transactions to complete, which ensures that executions are recoverable.
- ✓ There is no conflict between write operations of different transactions, because each transaction writes its own committed version of the objects it accesses.
- ✓ The rule in this ordering is

Tcmust not write objects that have been read by any Ti where Ti >Tc

Multi-version Timestamp ordering write rule

if (read timestamp of DmaxEarlier" Tc) perform write operation on a tentative version of D with write timestamp Tc else abort transaction Tc.

4.13 ATOMIC COMMIT PROTOCOL

- ➢ The atomicity property of transactions demands when a distributed transaction comes to an end, either all of its operations are carried out or none of them.
- ➢ In the case of a distributed transaction, the client has requested operations at more than one server.
- ➢ A transaction comes to an end when the client requests that it be committed or aborted.
- ➢ A simple way to complete the transaction in an atomic manner is for the coordinator to communicate the commit or abort request to all of the participants in the transaction and to keep on repeating the request until all of them have acknowledged that they have carried it out. This is **one phase atomic commit protocol.**

Two Phase Commit Protocol

- The **two-phase commit protocol** is designed to allow any participant to abort its part of a transaction.
- Due to the requirement for atomicity, if one part of a transaction is aborted, then the whole transaction must be aborted.
- In the first phase of the protocol, each participant votes for the transaction to be committed or aborted.
- Once a participant has voted to commit a transaction, it is not allowed to abort it.
- In the second phase of the protocol, every participant in the transaction carries out the joint decision.
- If any one participant votes to abort, then the decision must be to abort the transaction.
- If all the participants vote to commit, then the decision is to commit the transaction.
- The following are the operations in two phase commit protocol:
  - **canCommit?(trans)→Yes / No**: Call from coordinator to participant to ask whether it can commit a transaction. Participant replies with its vote.
  - **doCommit(trans):**Call from coordinator to participant to tell participant to commit its part of a transaction.
  - **doAbort(trans):** Call from coordinator to participant to tell participant to abort its part of a transaction.
  - **haveCommitted(trans, participant):** Call from participant to coordinator to confirm that it has committed the transaction.
  - **getDecision(trans) →Yes / No:** Call from participant to coordinator to ask for the decision on a transaction when it has voted Yes but has still had no reply after some delay. Used to recover from server crash or delayed messages

Two phase commit protocol

Phase 1 (voting phase):

1.The coordinator sends a canCommit? request to each of the participants in the transaction.

2. When a participant receives a canCommit? request it replies with its vote (Yes or No) to the coordinator. Before voting Yes, it prepares to commit by saving objects in permanent storage. If the vote is No, the participant aborts immediately.

Phase 2 (completion according to outcome of vote):

3. The coordinator collects the votes (including its own).

(a)If there are no failures and all the votes are Yes, the coordinator decides to commit the transaction and sends a doCommitrequest to each of the participants.

(b)Otherwise, the coordinator decides to abort the transaction and sends doAbortrequests to all participants that voted Yes.

4. Participants that voted Yes are waiting for a doCommitor doAbortrequest from the coordinator. When a participant receives one of these messages it acts accordingly and, in the case of commit, makes a have Committedcall as confirmation to the coordinator.

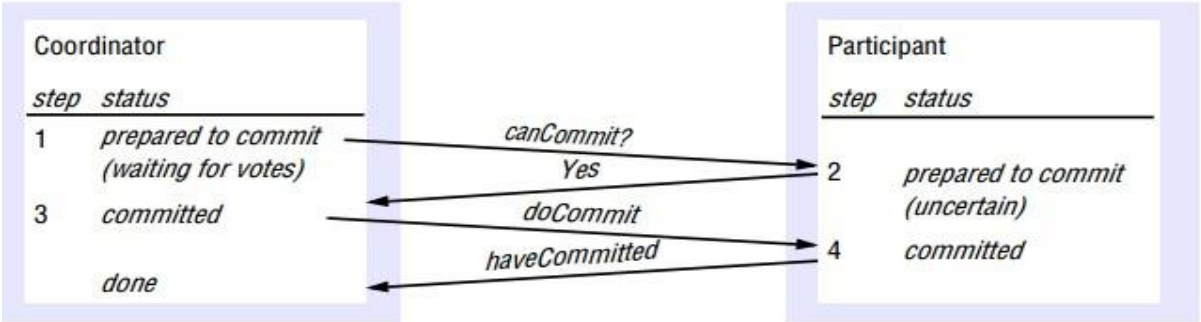


Fig 4.16: Two phase commit protocol

Two phase commit for nested transactions

- The outermost transaction in a set of nested transactions is called the top-level transaction.
- Transactions other than the top-level transaction are called sub-transactions.
- The following are the operations are allowed:
  - openSubTransaction(trans) →subTrans: Opens a new subtransaction whose parent is trans and returns a unique subtransaction identifier.
  - getStatus(trans)→committed, aborted, provisional: Asks the coordinator to report on the status of the transaction trans. Returns values representing one of the following: committed, aborted or provisional.
- When a sub-transaction completes, it makes an independent decision either to commit provisionally or to abort.
- A coordinator for a sub-transaction will provide an operation to open a sub-transaction, together with an operation enabling that coordinator to enquire whether its parent has yet committed or aborted.

Hierarchic Two phase commit protocol

- In this approach, the two-phase commit protocol becomes a multi-level nested protocol.
- The coordinator of the top-level transaction communicates with the coordinators of the sub-transactions for which it is the immediate parent.
- It sends canCommit? messages to each of the latter, which in turn pass them on to the coordinators of their child transactions.
- canCommit?(trans, subTrans) →Yes / No: Call from coordinator to coordinator of child sub-transaction to ask whether it can commit a sub-transaction subTrans. The first argument, trans, is the transaction identifier of the top-level transaction. Participant replies with its vote, Yes / No.

Flat Two phase commit protocol

- ✓ In this approach, the coordinator of the top-level transaction sends canCommit? messages to the coordinators of all of the sub-transactions in the provisional commit list.
- ✓ During the commit protocol, the participants refer to the transaction by its top-level TID.
- ✓ Each participant looks in its transaction list for any transaction or sub-transaction matching that TID.
- ✓ A participant can commit descendants of the top-level transaction unless they have aborted ancestors.
- ✓ When a participant receives a canCommit? request, it does the following:
- ✓ If the participant has any provisionally committed transactions that are descendants of the top-level transaction, trans, it:
  - checks that they do not have aborted ancestors in the abortList, then prepares to commit (by recording the transaction and its objects in permanent storage);
  - aborts those with aborted ancestors;
  - sends a Yes vote to the coordinator.
- ✓ If the participant does not have a provisionally committed descendent of the top level transaction, it must have failed since it performed the sub- transaction and it sends a No vote to the coordinator

4.14 DISTRIBUTED DEADLOCKS

- A cycle in the global wait-for graph (but not in any single local one) represents a distributed deadlock.
- A deadlock that is detected but is not really a deadlock is called a **phantom deadlock.**
- Two-phase locking prevents phantom deadlocks; autonomous aborts may cause phantom deadlocks.
- Permanent blocking of a set of processes that either compete for system resources or communicate with each other is deadlock.
- No node has complete and up-to-date knowledge of the entire distributed system. This is the cause of deadlocks.

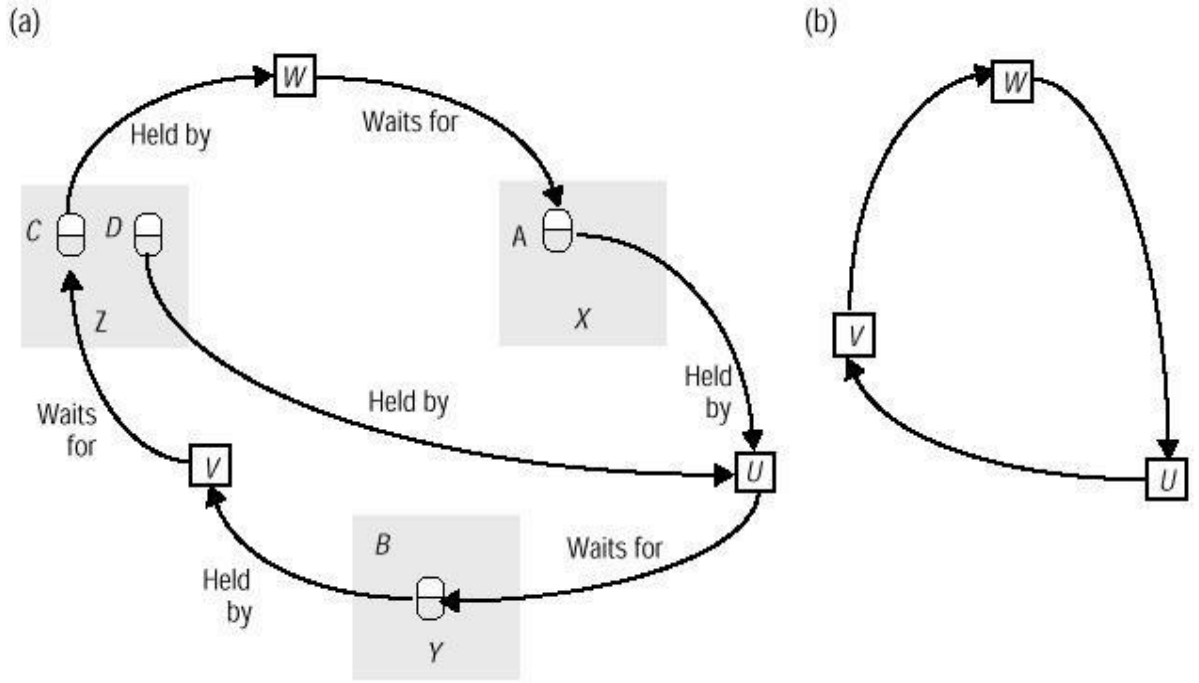
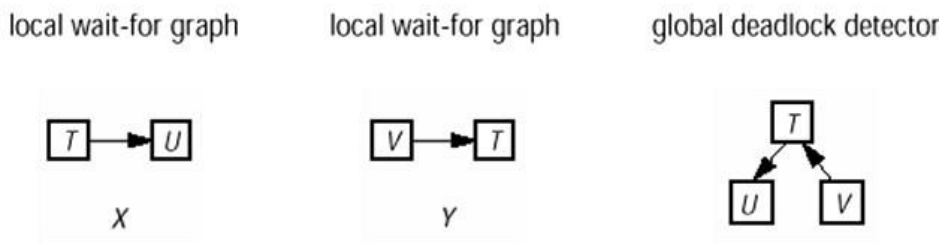


Fig 4.17: Distributed deadlocks and wait for graphs

Types of distributed deadlock

- **Resource deadlock :**Set of deadlocked processes, where each process waits for a resource held by another process (e.g., data object in a database, I/O resource on a server)
- **Communication deadlocks:** Set of deadlocked processes, where each process waits to receive messages (communication) from other processes in the set.

Local and Global Wait for Graphs



Edge Chasing

- When a server notes that a transaction T starts waiting for another transaction U, which is waiting to access a data item at another server, it sends a probe containing ⟨T→U⟩ to the server of the data item at which transaction U is blocked.
- **Detection:** receive probes and decide whether deadlock has occurred and whether to forward the probes.
- When a server receives a probe ⟨T→U⟩ and finds the transaction that U is waiting for, say V, is waiting for another data item elsewhere, a probe ⟨T→U→V⟩ is forwarded.
- **Detection:** receive probes and decide whether deadlock has occurred and whether to forward the probes.

  When a server receives a probe ⟨T→U⟩ and finds the transaction that U is waiting for, say V, is waiting for another data item elsewhere, a probe ⟨T→U→V⟩ is forwarded.
- **Resolution:** select a transaction in the cycle to abort

Transaction priorities

- ✓ Every transaction involved in a deadlock cycle can cause deadlock detection to be initiated.
- ✓ The effect of several transactions in a cycle initiating deadlock detection is that detection may happen at several different servers in the cycle, with the result that more than one transaction in the cycle is aborted.

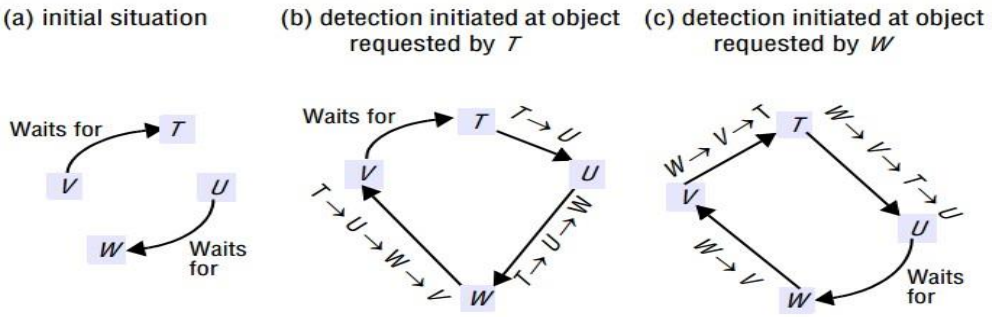


Fig 4.17: Initiated probes

- ✓ Consider transactions T, U, V and W, where U is waiting for Wand V is waiting for T.
- ✓ At about the same time, T requests the object held by U and W requests the object held by V.
- ✓ Two separate probes, <T →U > and <W →V >, are initiated by the servers of these objects and are circulated until deadlocks are detected by each of the servers.
- ✓ The cycle is <T →U→W →V →T>and, the cycle is <W →V →T →U →W >.
- ✓ In order to ensure that only one transaction in a cycle is aborted, transactions are given priorities in such a way that all transactions are totally ordered.
- ✓ In order to ensure that only one transaction in a cycle is aborted, transactions are given priorities in such a way that all transactions are totally ordered.
- ✓ The problem is that the order in which transactions start waiting can determine whether or not a deadlock will be detected.
- ✓ The above pitfall can be avoided by using as scheme in which coordinators save copies of all the probes received on behalf of each transaction in a **probe queue.**
- ✓ When a transaction starts waiting for an object, it forwards the probes in its queue to the server of the object, which propagates the probes on down hill routes.

4.14 REPLICATION

Replication in distributed systems enhances the performance, availability and fault tolerance. The general requirement includes:

- Replication transparency
- Consistency

4.14.1 System Model

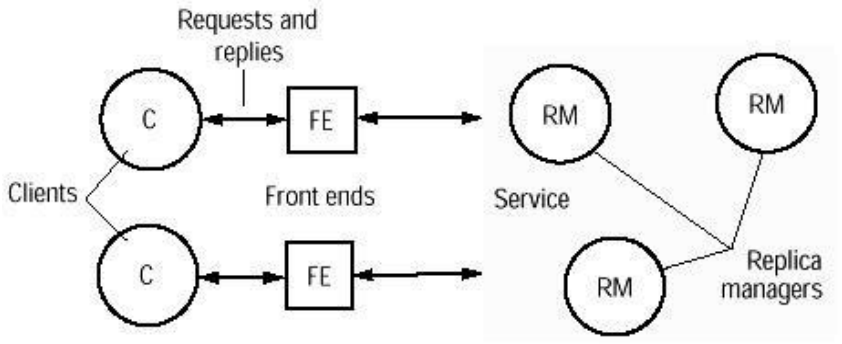


Fig 4.18: Architecture for replication management

- ➢ The data in a system consist of a collection of items called objects.

- An object could be a file, say, or a Java object.
- But each such logical object is implemented by a collection of physical copies called replicas.
- The replicas are physical objects, each stored at a single computer, with data and behavior that are tied to some degree of consistency by the system‟s operation.
- The system models include replica managers which are components that contain the replicas on a given computer and perform operations upon them directly.
- Each client‟s requests are first handled by a component called a front end.
- The role of the front end is to communicate by message passing with one or more of the replica managers, rather than forcing the client to do this itself explicitly.
- It is the vehicle for making replication transparent.
- A front end may be implemented in the client‟s address space, or it may be a separate process.

Phases in request processing:

- **Issuance of request:** The front end issues the request to one or more replica managers:
  – either the front end communicates with a single replica manager, which in turn communicates with other replica managers

  – or the front end multicasts the request to the replica managers.
- **Coordination:**
  - The replica managers coordinate in preparation for executing therequest consistently.
  - They agree, if necessary at this stage, on whether the request is to be applied.
  - They also decide on the ordering of this request relative to others.
  - The types of ordering includes: FIFO ordering, casual ordering and total ordering.
- **Execution:** The replica managers execute the request – perhaps tentatively: that is, in such a way that they can undo its effects later.
- **Agreement:** The replica managers reach consensus on the effect of the request – if any – that will be committed.
- **Response:** One or more replica managers respond to the front end.

4.15 CASE STUDY: CODA

Coda is a distributed file system. Coda has been developed at Carnegie Mellon University (CMU) in the 1990s, and is now integrated with a number of popular UNIX-based operating systems such as Linux.

Features of Coda File System (CFS):

- CFS main goal is to achieve high availability.
- It has advanced caching schemes.
- It provide transparency

Architecture of Coda

- The clients cache entire files locally.
- Cache coherence is maintained by the use of callbacks. Clients dynamically find files on server and cache location information.
- For security, token-based authentication and end-to-end encryption is used.

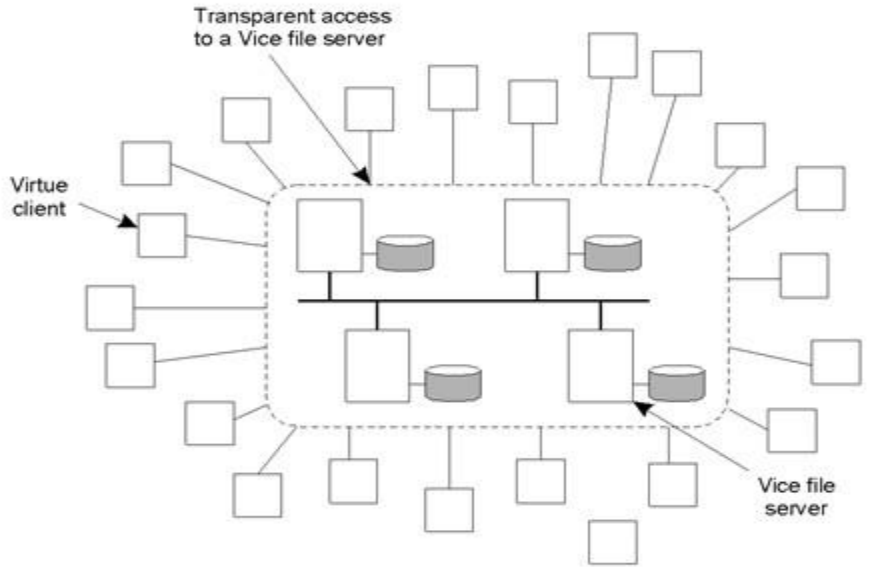


Fig 4.18: Coda file systems

- It is a principle of the design of Coda that the copies of files residing on servers are more reliable than those residing in the caches of clients.
- It is possible to construct a file system that relies entirely on cached copies of files in client.
- But such systems will have poor QOS.

- The Coda servers exist to provide the necessary quality of service.
- The copies of files residing in client caches are regarded as useful only as long as their currency can be revalidated against the copies in server.
- Revalidation occurs when disconnected operation ceases and the cached files are reintegrated with those in the servers.

Communication in Coda

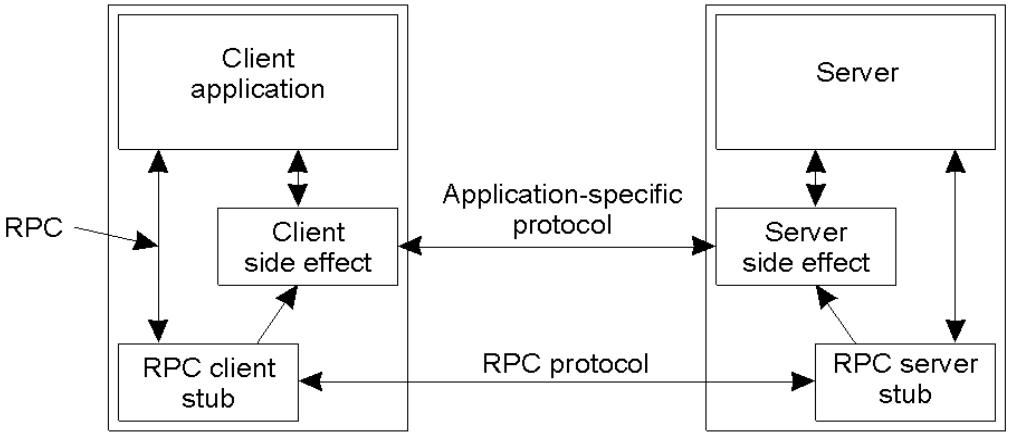


Fig 4.19: Communication in Coda

- Coda uses RPC2: a sophisticated reliable RPC system.
- The RPC2 start a new thread for each request, server periodically informs client it is still working on the request.
- RPC2 is useful for video streaming.
- RPC2 also has multicast support
- A side effect is a mechanism by which the client and server can communicate using an application-specific protocol.
- Coda uses side effect mechanism.
- Coda servers allow clients to cache whole files.
- Modifications by other clients are notified through invalidation messages require multicast RPC. The modifications can be done in any one of the following ways:
  - Sending an invalidation message one at a time
  - Sending invalidation messages in parallel

Processes in Coda

Coda ts are known as Venus pr e and operate entirely in us

Caching in

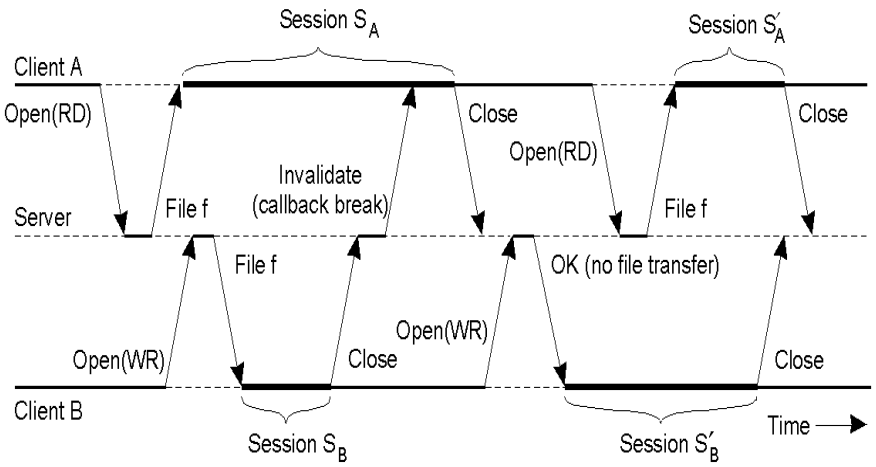


Fig 4.20: Caching in Coda

and provide

Server Replication in Coda

- ✓ The basic unit of replication is termed as volume.
- ✓ The Volume Storage Group (VSG) is set of servers that have a copy of a volume.
- ✓ The Accessible Volume Storage Group (AVSG) is set of servers in VSG that the client can contact .
- ✓ The Coda uses vector versioning:
  - ➢ One entry for each server in VSG
  - ➢ When file updated, corresponding version in AVSG is updated

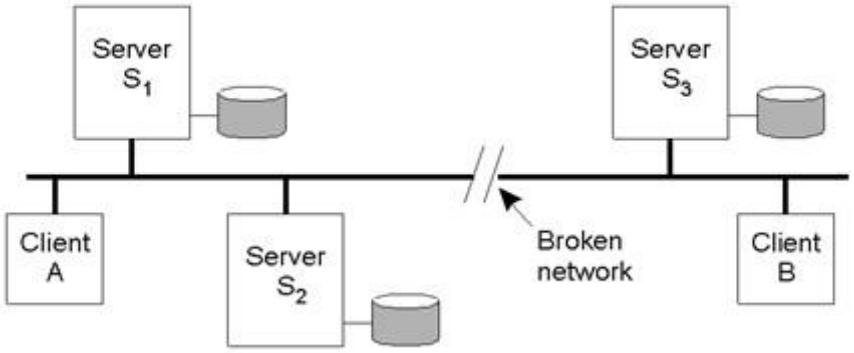


Fig 4.21: Server Replication

- ✓ In the above figure, the versioning vector at the time of partition is :[1,1,1]
- ✓ Client A updates file : versioning vector in its partition: [2,2,1]
- ✓ Client B updates file: versioning vector in its partition: [1,1,2]
- ✓ After the partition is repaired, compare versioning vectors. There will be a conflict.

Fault tolerance

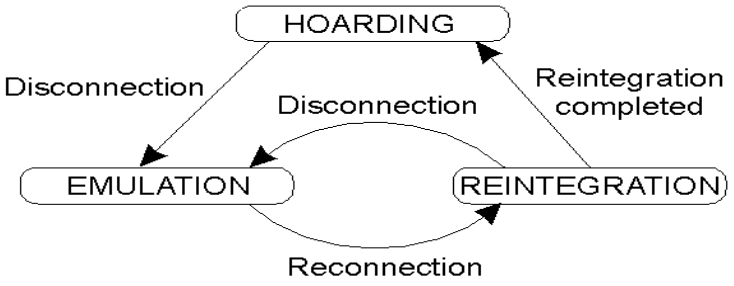


Fig 4.22: Fault Tolerance

The following are the fault tolerating actions:

- **HOARDING**: File cache in advance with all files that will be accessed when disconnected
- **EMULATION:** when disconnected, behavior of server emulated at client
- **REINTEGRATION:** transfer updates to server; resolves conflicts

REVIEW QUESTIONS

PART – A

1. **Define clock skew.**

   Clock skew is defined as the difference between the times on two clocks.

2. **Define clock drift.**

   Clock drift is the count time at different rates.

3. **What is Clock drift rate ?**

   Clock drift rate is the difference in precision between a prefect reference clock and a physical clock.

4. **What is External synchronization?**

   - This method synchronize the process‟s clock with an authoritative external reference clock S(t) by limiting skew to a delay bound D > 0 - |S(t) - Ci(t) | < D for all t.
   - For example, synchronization with a UTC (Coordinated Universal Time)source.

5. **What is Internal synchronization?**

   - Synchronize the local clocks within a distributed system to disagree by not more than a delay bound D > 0, without necessarily achieving external synchronization - |Ci(t) - Cj(t)| < D for all i, j, t n
   - For a system with external synchronization bound of D, the internal synchronization is bounded by 2D.

6. **Give the types of clocks.**

   Two types of clocks are used:

   - Logical clocks : to provide consistent event ordering
   - Physical clocks : clocks whose values must not deviate from the real time by more than a certain amount.

7. **What are the techniques are used to synchronize clocks?**

   - ✓ time stamps of real-time clocks
   - ✓ message passing
   - ✓ round-trip time (local measurement)

8. List the algorithms that provides clock synchronization.

- Cristian‟s algorithm
- Berkeley algorithm
- Network time protocol (Internet)

9. Give the working of Berkley's algorithm.

The time daemon asks all the other machines for their clock values. The machines answer the request. The time daemon tells everyone how to adjust their clock.

10. What is NTP?

The Network Time Protocol defines architecture for a time service and a protocol to distribute time information over the Internet.

11. Give the working of Procedure call and symmetric modes:

- All messages carry timing history information.
- The history includes the local timestamps of send and receive of the previous NTP message and the local timestamp of send of this message
- For each pair i of messages (m, m‟) exchanged between two servers the following values are being computed
  - $offset o_i$: estimate for the actual offset between two clocks
  - delay d_i : true total transmission time for the pair of messages.

12. Define casual ordering.

The partial ordering obtained by generalizing the relationship between two process is called as happened-before relation or causal ordering or potential causal ordering.

13. Define logical clock.

A Lamport logical clock is a monotonically increasing software counter, whose value need bear no particular relationship to any physical clock.

14. What is global state?

The global state of a distributed system consists of the local state of each process, together with the messages that are currently in transit, that is, that have been sent but not delivered.

15. When do you call an object to be a garbage?

An object is considered to be garbage if there are no longer any references to it anywhere in the distributed system.

16. Define distributed deadlock.

A distributed deadlock occurs when each of a collection of processes waits for another process to send it a message, and where there is a cycle in the graph of this „waits-for" relationship.

17. What is distributed snapshot?

Distributed Snapshot represents a state in which the distributed system might have been in. A snapshot of the system is a single configuration of the system.

18. What is consistent cut?

A consistent global state is one that corresponds to a consistent cut.

19. Define run.

A run is a total ordering of all the events in a global history that is consistent with each local history"s ordering.

20. What is linearization?

A linearization or consistent run is an ordering of the events in a global history that is consistent with this happened-before relation o on H.

21. What is global state predicate?

A global state predicate is a function that maps from the set of global states of processes in the system.

22. What are the features of the unreliable failure detectors?

- unsuspected or suspected (i.e.) there can be no evidence of failure
- each process sends ``alive" message to everyone else
- not receiving ``alive" message after timeout
- This is present in most practical systems

23. What are the features of the reliable failure detectors?

- Unsuspected or failure
- They are present in synchronous system

24. Define distributed mutual exclusion.

Distributed mutual exclusion provide critical region in a distributed environment.

25. What are the requirements for Mutual Exclusion (ME)?

[ME1] safety: only one process at a time

[ME2] liveness: eventually enter or exit

[ME3] happened-before ordering: ordering of enter() is the same as HB ordering

26. What are the criteria for performance measures?

Bandwidth consumption, which is proportional to the number of messages sent in each entry and exit operations.

The client delay incurred by a process at each entry and exit operation.

Throughput of the system: Rate at which the collection of processes as a whole can access the critical section.

27. What is ring based algorithm?

This provides a simplest way to arrange mutual exclusion between N processes without requiring an additional process is arrange them in a logical ring.

28. What is mutual synchronisation?

This exploits mutual exclusion between N peer processes based upon multicast. 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.

29. What is Maekawa's Voting Algorithm?

In this algorithm, it is not necessary for all of its peers to grant access. Only need to obtain permission to enter from subsets of their peers, as long as the subsets used by any two processes overlap.

30. What is election algorithm?

An algorithm for choosing a unique process to play a particular role is called an election algorithm.

31. What is Ring based Election Algorithm?

- All the processes arranged in a logical ring.
- Each process has a communication channel to the next process.
- All messages are sent clockwise around the ring.
- Assume that no failures occur, and system is asynchronous.
- The ultimate goal is to elect a single process coordinator which has the largest identifier

32. What is bully algorithm?

This algorithm allows process to crash during an election, although it assumes the message delivery between processes is reliable.

33. What are the messages in Bully algorithm?

There are three types of messages:

a. Election message: This is sent to announce an election message. A process begins an election when it notices, through timeouts, that the coordinator has failed. T=2Ttrans+Tprocess From the time of sending

b. Answermessage: This is sent in response to an election message.

c. Coordinatormessage: This is sent to announce the identity of the elected process.

34. Define transaction.

A Transaction defines a sequence of server operations that is guaranteed to be atomic in the presence of multiple clients and server crash.

35. List the methods to ensure serializability.

There are three ways to ensure serializability:

- Locking
- Timestamp ordering
- Optimistic concurrency control

36. Give the advantages of nested transactions.

- Sub- transactions at same level can run concurrently.
- Sub- transactions can commit or abort independently.

37. Give the rules for committing of nested transactions.

- A transaction may commit or abort only after its child transactions have completed.
- When a sub-transaction completes, it makes an independent decision either to commit provisionally or to abort. Its decision to abort is final.
- When a parent aborts, all of its sub-transactions are aborted.
- When a sub -transaction aborts, the parent can decide whether to abort or not.
- If the top-level transaction commits, then all of the sub-transactions that have provisionally committed can commit too, provided that none of their ancestors has aborted.

38. What are the information held by the locks?

Each instance of Lock maintains the following information in its instance variables:

* the identifier of the locked object
* the transaction identifiers of the transactions that currently hold the lock
* a lock type

39. What is two phase locking?

The basic two-phase locking (2PL) protocol states:

- ✓ A transaction T must hold a lock on an item x in the appropriate mode before T accesses x.
- ✓ If a conflicting lock on x is being held by another transaction, T waits.
- ✓ Once T releases a lock, it cannot obtain any other lock subsequently.

40. Define deadlock.

Deadlock is a state in which each member of a group of transactions is waiting for some other member to release a lock.

41. Give the disadvantages of locking and serialization.

- ✓ Lock maintenance represents an overhead that is not present in systems that do not support concurrent access to shared data. Locking sometimes are only needed for some cases with low probabilities.
- ✓ The use of lock can result in deadlock. Deadlock prevention reduces concurrency severely. The use of timeout and deadlock detection is not ideal for interactive programs.
- ✓ To avoid cascading aborts, locks cannot be released until the end of the transaction. This may reduce the potential for concurrency.

42. What is multi version time stamp ordering?

In multi-version timestamp ordering, a list of old committed versions as well as tentative versions is kept for each object. This list represents the history of the values of the object. The benefit of using g multiple versions is that read operations that arrive too late need not be rejected.

43. What is Hierarchic Two phase commit protocol?

* In this approach, the two-phase commit protocol becomes a multi-level nested protocol.
* The coordinator of the top-level transaction communicates with the coordinators of the sub-transactions for which it is the immediate parent.

44. What is Flat Two phase commit protocol?

- ✓ In this approach, the coordinator of the top-level transaction sends canCommit? messages to the coordinators of all of the sub-transactions in the provisional commit list.
- ✓ During the commit protocol, the participants refer to the transaction by its top-level TID.
- ✓ Each participant looks in its transaction list for any transaction or sub-transaction matching that TID.

45. Define phantom deadlock.

A deadlock that is detected but is not really a deadlock is called a phantom deadlock.

46. What is replication?

Replication in distributed systems enhances the performance, availability and fault tolerance.

47. What is Coda?

Coda is a distributed file system. Coda has been developed at Carnegie Mellon University (CMU) in the 1990s, and is now integrated with a number of popular UNIX-based operating systems such as Linux.

48. What are the fault tolerating actions?

- HOARDING: File cache in advance with all files that will be accessed when disconnected
- EMULATION: when disconnected, behaviour of server emulated at client
- REINTEGRATION: transfer updates to server; resolves conflicts

PART - B

1. Explain the clocking in detail.
2. Describe Cristian‟s algorithm.
3. Write about Berkeley algorithm
4. Brief about NTP.
5. Explain about logical time and logical clocks.
6. Describe global states.

7. Brief about distributed mutual exclusion.
8. Write in detail about election algorithms.
9. Explain transactions and concurrency control.
10. Describe nested transaction and its issues.
11. What is optimistic concurrency control?
12. Write in detail about timestamp ordering.
13. Describe atomic commit protocol.
14. Explain distributed deadlocks.
15. Describe replication.
16. Write about Coda.

5

PROCESS AND RESOURCE MANAGEMENT

5.1 PROCESS MANAGEMENT

Process Management involves the activities aimed at defining a process, establishing responsibilities, evaluating process performance, and identifying opportunities for improvement.

5.1.1 Process Migration

- Process migration involves the transfer of sufficient amount of the state of a process from one computer to another.

Process migration is the act of transferring process between two machines.

- The process executes on the target machine.
- The aim for migration is to move processes from heavily loaded to lightly load systems.
- This enhances the performance communication.
- Processes that interact intensively can be moved to the same node to reduce communications cost.
- The hope of process migration is that it will allow for better system-wide utilization of resources.
- The long-running process may be moved because the machine it is running on will be down.
- Process can take advantage of unique hardware or software capabilities.

- Process migration enables dynamic load distribution, fault resilience, eased system administration, and data access locality.
- After migration, the process on the source system and create it on the target system.
- The process image and process control block and any links must be moved.

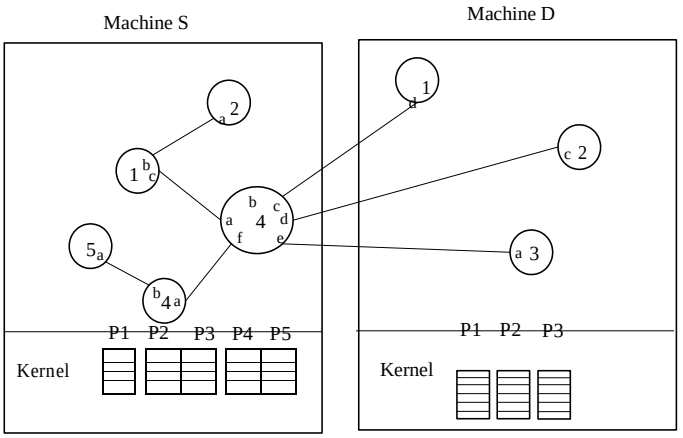


Fig 5.1 Before Migration

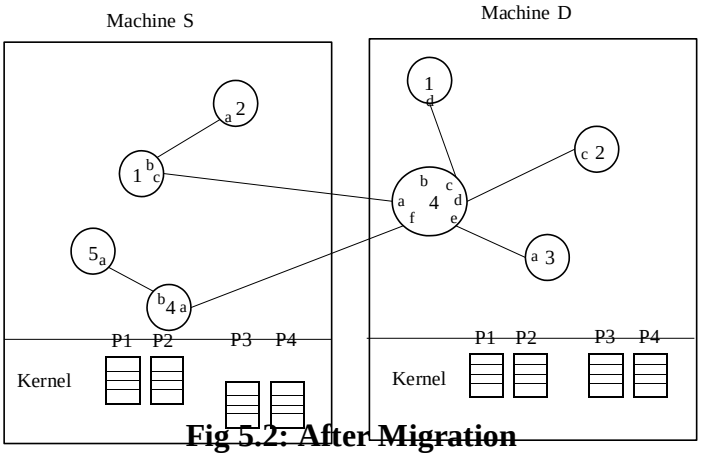


Fig 5.2: After Migration

Issues in migration:

The following questions must be addressed in process migration:

- Who initiates the migration?
- What is involved in a Migration?
- What portion of the process is migrated?
- What happens to outstanding messages and signals?

Moving PCBs:

- The movement of the process control block is straightforward and simple.
- The difficulty, from a performance point of view, concerns the process address space and any open files assigned to the process.
- There are several strategies for moving the address space and data including:
  - Eager (All):
    - This transfers the entire address space.
    - No trace of process is left behind in the original system.
    - If address space is large and if the process does not need most of it, then this approach is unnecessarily expensive.
    - Implementations that provide a checkpoint/restart facility are likely to use this approach, because it is simpler to do the check-pointing and restarting if all of the address space is localized.
  - Precopy
    - In this method, the process continues to execute on the source node while the address space is copied to the target node.
    - The pages modified on the source during the precopy operation have to be copied a second time.
    - This strategy reduces the time that a process is frozen and cannot execute during migration.
  - Eager (dirty)
    - The transfer involves only the portion of the address space that is in main memory and has been modified.
    - Any additional blocks of the virtual address space are transferred on demand.
    - The source machine is involved throughout the life of the process.
  - Copy-on-reference
    - In this method, the pages are only brought over when referenced.
    - This has the lowest initial cost of process migration.

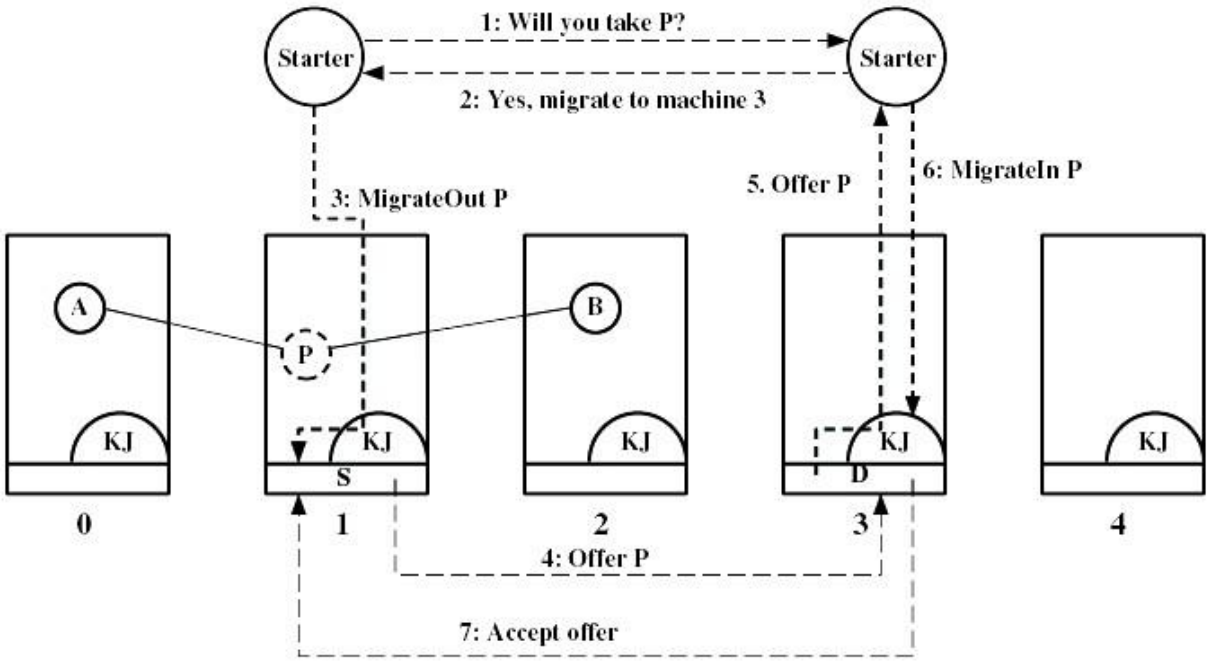


Fig 5.3: Negotiation in Migration process

Eviction

- Sometimes the destination system may refuse to accept the migration of a process to itself.
- If a workstation is idle, process may have been migrated to it.
- Once the workstation is active, it may be necessary to evict (recover) the migrated processes to provide adequate response time.

Distributed Global States

- The operating system cannot know the current state of all process in the distributed system.
- A process can only know the current state of all processes on the local system.
- Also the remote processes only know state information that is received by messages.
- This message helps to know about the state in the past.

5.2 THREADS

A thread is the entity within a process that can be scheduled for execution. All threads of a process share its virtual address space and system resources.

Each thread maintains exception handlers, a scheduling priority, thread local storage, a unique thread identifier, and a set of structures the system will use to save the thread context until it is scheduled. The ***thread context*** includes the thread's set of machine registers, the kernel stack, a thread environment block, and a user stack in the address space of the thread's process. Threads can also have their own security context, which can be used for impersonating clients.

Because threads have some of the properties of processes, they are sometimes called lightweight processes. In a process, threads allow multiple executions of streams. A process can be single threaded or multi- threaded.

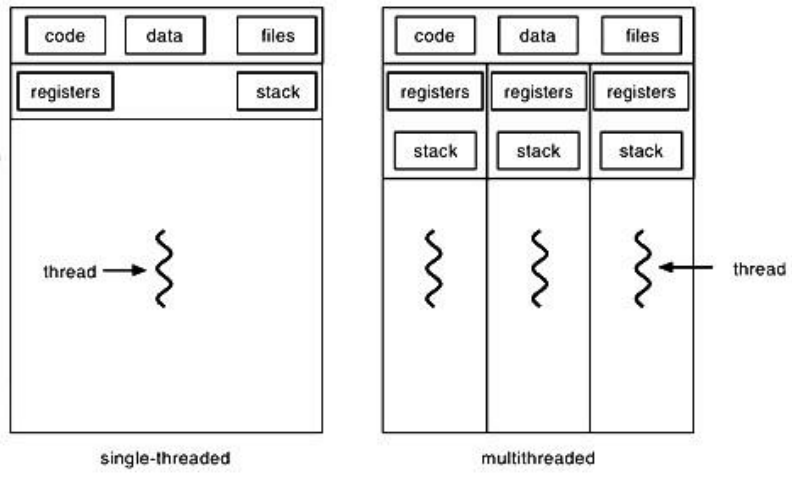


Fig 5.4: Threads

Threads provide better responsiveness, resource sharing, economy and better utilization of multiprocessor architectures.

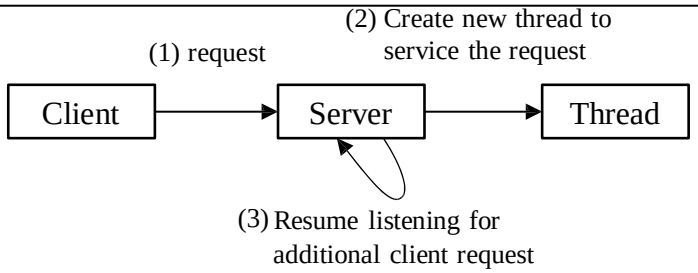


Fig 5.5: Multithreaded Architectures

Thread management done by user-level threads library. The actions in thread management involve:

- ❖ Creation and destruction of threads
- ❖ Message passing or data sharing between threads
- ❖ Scheduling threads
- ❖ saving/restoring threads contexts

5.2.1 Multithreading Models

There are basically three multithreading models based on how the user and kernel threads map into each other:

- ✓ Many to one
- ✓ One to one
- ✓ Many to many

Many to one Model

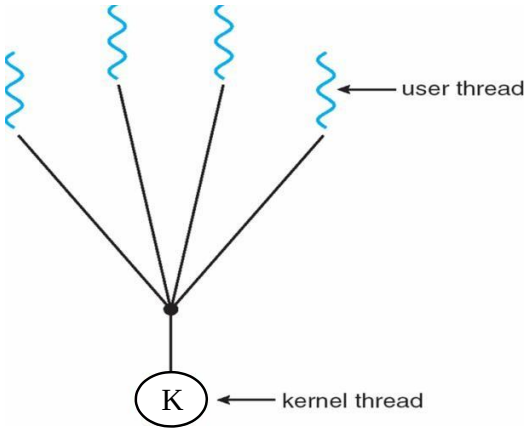


Fig 5.6: Many to one Model

In this model many user-level threads mapped to single kernel thread. Examples are Solaris Green Threads and GNU Portable Threads.

One to One Model

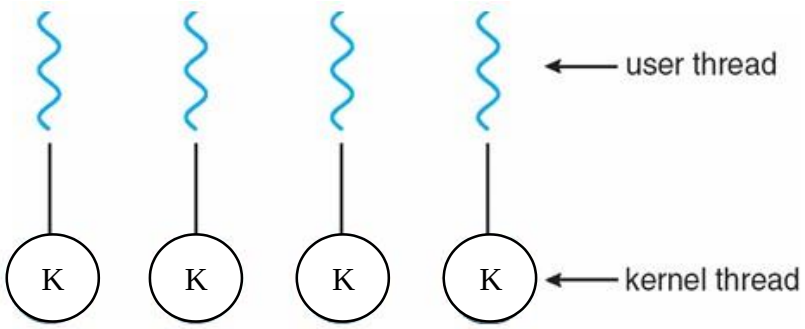


Fig 5.7: One to One Model

In this model each user-level thread maps to kernel thread. The disadvantage of this model is creating a user thread requires creating a kernel thread. Example are threads in Windows NT/XP/2000, Linux, Solaris 9 and later.

Many to many model

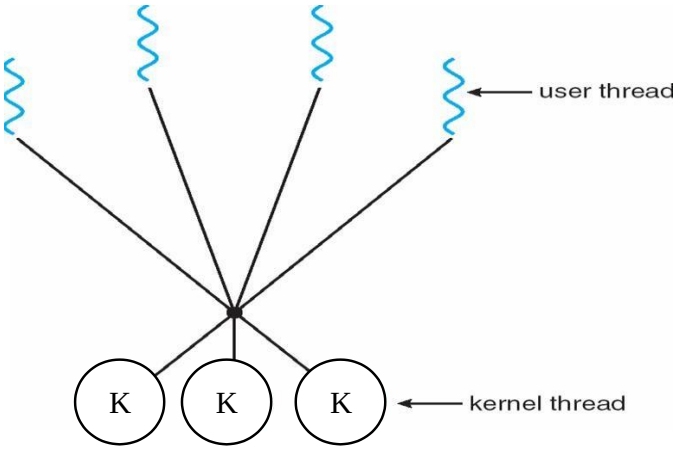


Fig 5.8: Many to many model

This model allows many user level threads to be mapped to many kernel threads. This also allows the operating system to create a sufficient number of kernel threads. Examples includes Solaris prior to version 9 and Windows NT/2000 with the *ThreadFiber* package.

Two Level Model

This is similar to Many to Many model, except that it allows a user thread to be bound to a kernel thread. Examples are threads in IRIX, HP-UX, Tru64 UNIX, Solaris 8 and earlier.

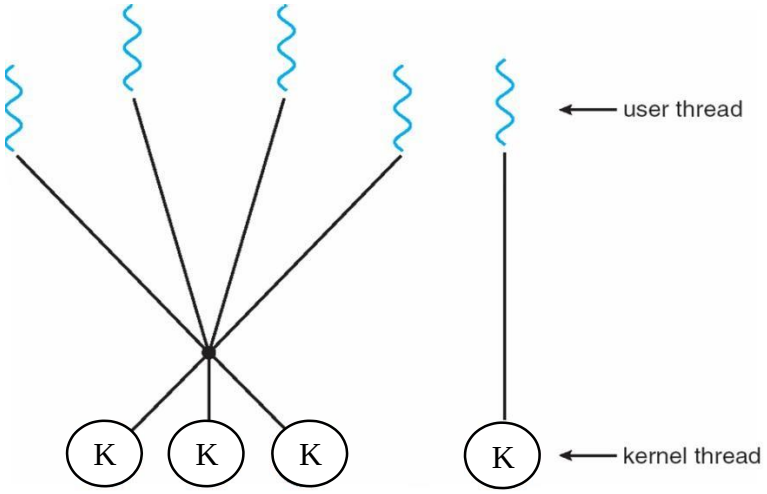


Fig 5.9: Two Level Model

5.2.2 Threading Issues

The following are the issues in threads:

- Semantics of fork() and exec() system calls

  The fork() call may duplicate either the calling thread or all threads.

- Thread cancellation
  - Terminating a thread before it has finished.
  - There are two general approaches:
    - ↑ Asynchronous cancellation terminates the target thread immediately
    - ↑ Deferred cancellation allows the target thread to periodically check if it should be cancelled
- Signal handling
  - Signals are used in UNIX systems to notify a process that a particular event has occurred
  - A signal handler is used to process signals
  - Signal is generated by particular event
  - Signal is delivered to a process
  - Signal is handled
  - The signal can be delivered :
    - ↑ to the thread to which the signal applies

- ↑ to every thread in the process
- ↑ to certain threads in the process

– Assign a specific thread to receive all signals for the process.

- Thread pools
  - Create a number of threads in a pool where they await work.
  - This is slightly faster to service a request with an existing thread than create a new thread
  - This allows the number of threads in the application(s) to be bound to the size of the pool.
- Thread specific data
  - This allows each thread to have its own copy of data
  - This is useful when you do not have control over the thread creation process (i.e., when using a thread pool)
- Scheduler activations
  - The Many to Many models require communication to maintain the appropriate number of kernel threads allocated to the application.
  - The scheduler activations provide upcalls: a communication mechanism from the kernel to the thread library.
  - This communication allows an application to maintain the correct number kernel threads.

5.2.3 Thread Implementation

The various thread implementations are listed below:

❖ **PThreads**

- This is a POSIX standard (IEEE 1003.1c).
- This is a API for thread creation and synchronization and specifies the behavior of the thread library.
- pthreads is an Object Orientated API that allows user-land multi-threading in PHP.
- It includes all the tools needed to create multi-threaded applications targeted at the Web or the Console.

- PHP applications can create, read, write, execute and synchronize with Threads, Workers and Threaded objects.
- **A Threaded Object:**
  - A Threaded Object forms the basis of the functionality that allows pthreads to operate.
  - It exposes synchronization methods and some useful interfaces for the programmer.
- **Thread:**
  - The user can implement a Thread by extending the Thread declaration provided by pthreads implementing the run method.
  - Any members of the thread can be written and read by any context with a reference to the Thread.
  - Any context can be executed in any public and protected methods.
  - The run method of the implementation is executed in a separate thread when the start method of the implementation is called from the context that created it.
  - Only the context that creates a thread can start and join with it.
- **Worker Object:**
  - A Worker Thread has a persistent state, and will be invoked by start().
  - The life of worker thread is till the calling object goes out of scope, or is explicitly shutdown.
  - Any context with a reference can stack objects onto the Worker, which will be executed by the Worker in a separate Thread.
  - The run method of a Worker is executed before any objects on the stack, such that it can initialize resources that the objects to come may need.
- **Pool:**
  - A Pool of Worker threads can be used to distribute Threaded objects among Workers.
  - The Pool class included implements this functionality and takes care of referencing in a sane manner.
  - The Pool implementation is the easiest and most efficient way of using multiple threads.

- **Synchronization:**
  - All of the objects that pthreads creates have built in synchronization in the (familiar to java programmers) form of ::wait and ::notify.
  - Calling ::wait on an object will cause the context to wait for another context to call ::notify on the same object.
  - This allows for powerful synchronization between Threaded Objects in PHP.
- **Method Modifiers:**
  - The protected methods of Threaded Objects are protected by pthreads, such that only one context may call that method at a time.
  - The private methods of Threaded Objects can only be called from within the Threaded Object during execution.
- **Data Storage:**
  - Any data type that can be serialized can be used as a member of a Threaded object, it can be read and written from any context with a reference to the Threaded Object.
  - Not every type of data is stored serially, basic types are stored in their true form. Complex types, Arrays, and Objects that are not Threaded are stored serially; they can be read and written to the Threaded Object from any context with a reference.
  - With the exception of Threaded Objects any reference used to set a member of a Threaded Object is separated from the reference in the Threaded Object; the same data can be read directly from the Threaded Object at any time by any context with a reference to the Threaded Object.
- **Static Members:**
  - When a new context is created , they are generally copied, but resources and objects with internal state are nullified .
  - This allows them to function as a kind of thread local storage.
  - Allowing the new context to initiate a connection in the same way as the context that created it, storing the connection in the same place without affecting the original context.
  - These threads are common in UNIX operating systems (Solaris, Linux, Mac OS X).

❖ **Windows Threads**

- Microsoft Windows supports preemptive multitasking, which creates the effect of simultaneous execution of multiple threads from multiple processes.
- On a multiprocessor computer, the system can simultaneously execute as many threads as there are processors on the computer.
- A **job object** allows groups of processes to be managed as a unit.
- Job objects are namable, securable, sharable objects that control attributes of the processes associated with them.
- Operations performed on the job object affect all processes associated with the job object.
- An application can use the **thread pool** to reduce the number of application threads and provide management of the worker threads.
- Applications can queue work items, associate work with waitable handles, automatically queue based on a timer, and bind with I/O.
- **User-mode scheduling (UMS)** is a lightweight mechanism that applications can use to schedule their own threads.
- An application can switch between UMS threads in user mode without involving the system scheduler and regain control of the processor if a UMS thread blocks in the kernel.
- Each UMS thread has its own thread context instead of sharing the thread context of a single thread.
- The ability to switch between threads in user mode makes UMS more efficient than thread pools for short-duration work items that require few system calls.
- A **fiber** is a unit of execution that must be manually scheduled by the application.
- Fibers run in the context of the threads that schedule them.
- Each thread can schedule multiple fibers.
- In general, fibers do not provide advantages over a well-designed multithreaded application.
- However, using fibers can make it easier to port applications that were designed to schedule their own threads.

❖ **Java Threads**

- Java is a multi threaded programming language which means we can develop multi threaded program using Java.
- A multi threaded program contains two or more parts that can run concurrently and each part can handle different task at the same time making optimal use of the available resources especially when your computer has multiple CPUs.
- Java threads may be created by: Extending Thread class or by implementing the Runnable interface.
- Java threads are managed by the JVM.

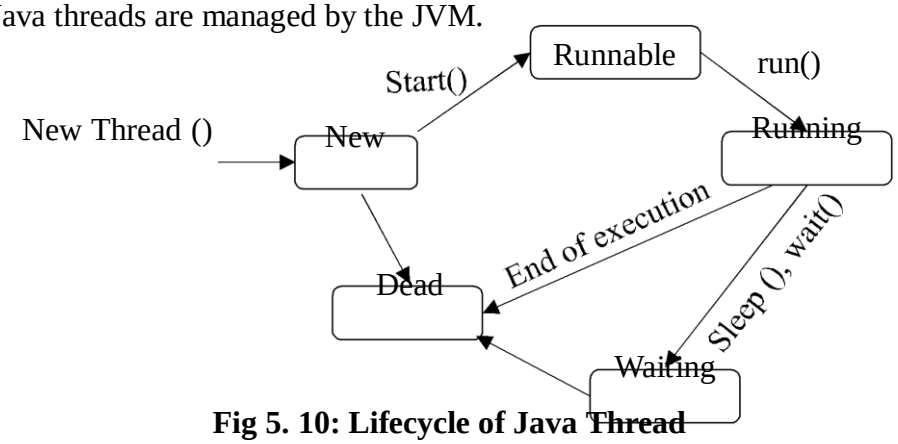


Fig 5. 10: Lifecycle of Java Thread

- Thread class provide constructors and methods to create and perform operations on a thread.
- Thread class extends Object class and implements Runnable interface.
- The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread.
- Runnable interface have only one method named run().

5.3 RESOURCE MANAGEMENT IN DISTRIBUTED SYSTEMS

✓ Distributed systems contain a set of resources interconnected by a network.

✓ Processes are migrated to fulfill their resource requirements.

✓ The Resource manager is responsible to control the assignment of resources to processes.

✓ The resources can be logical (shared file) or physical (CPU) resource.

✓ Scheduling is the way in which processors are assigned to run on the available resources.

- ✓ In a distributed computing system, the scheduling of various modules on particular processing nodes may be preceded by appropriate allocation of modules of the different tasks to various processing nodes and then only the appropriate execution characteristic can be obtained.
- ✓ The task allocation becomes the important most and major activity in the task scheduling within the operating system of a DCS.

5.3.1 Features of Scheduling algorithms

The following are the desired features of scheduling algorithms:

- **General purpose**
  - ✓ A scheduling approach should make few assumptions about and have few restrictions to the types of applications that can be executed.
  - ✓ Interactive jobs, distributed and parallel applications, as wellas non-interactive batch jobs, should all be supported with good performance.
  - ✓ This property is a straightforward one, but to some extent difficult to achieve.
  - ✓ Because different kinds of jobs have different attributes, their requirements to the scheduler may contradict.
  - ✓ To achieve the general purpose, a tradeoff may have to be made.
- **Efficiency:**
  - ✓ It has two meanings: one is that it should improve the performance of scheduled jobs as much as possible; the other is that the scheduling should incur reasonably low overhead so that it would not counter attack the benefits.
- **Fairness:**
  - ✓ Sharing resources among users raises new challenges in guaranteeing that each user obtains his/her fair share when demand is heavy is fairness.
  - ✓ In a distributed system, this problem could be exacerbated suchthat one user consumes the entire system.
  - ✓ There are many mature strategies to achieve fairness on a single node.
- **Dynamic:**
  - ✓ The algorithms employed to decide where to process a task should respond to load changes, and exploit the full extent of the resources available.

- **Transparency:**
  - ✓ The behavior and result of a task's execution should not be affected by the host(s) on which it executes.
  - ✓ In particular, there should be no difference between local and remote execution.
  - ✓ No user effort should be required in deciding where to execute a task or in initiating remote execution; a user should not even be aware of remote processing.
  - ✓ Further, the applications should not be changed greatly.
  - ✓ It is undesirable to have to modify the application programs in order to execute them in the system.
- **Scalability**
  - ✓ A scheduling algorithm should scale well as the number of nodes increases.
  - ✓ An algorithm that makes scheduling decisions by first inquiring the workload from all the nodes and then selecting the most lightly loaded node has poor scalability.
  - ✓ This will work fine only when there are few nodes in the system.
  - ✓ This is because the inquirer receives a flood of replies almost simultaneously, and the time required to process the reply messages for making a node selection is too long as the number of nodes (N) increase.
  - ✓ Also the network traffic quickly consumes network bandwidth.
  - ✓ A simple approach is to probe only m of N nodes for selecting a node.
- **Fault tolerance**
  - ✓ A good scheduling algorithm should not be disabled by the crash of one or more nodes of the system.
  - ✓ Also, if the nodes are partitioned into two or more groups due to link failures, the algorithm should be capable of functioning properly for the nodes within a group.
  - ✓ Algorithms that have decentralized decision making capability and consider only available nodes in their decision making have better fault tolerance capability.

- **Quick decision making capability**
  - ✓ Heuristic methods requiring less computational efforts (and hence less time) while providing near-optimal results are preferable to exhaustive (optimal) solution methods.
- **Balanced system performance and scheduling overhead**
  - ✓ Algorithms that provide near-optimal system performance with a minimum of global state information (such as CPU load) gathering overhead are desirable.
  - ✓ This is because the overhead increases as the amount of global state information collected increases.
  - ✓ This is because the usefulness of that information is decreased due to both the aging of the information being gathered and the low scheduling frequency as a result of the cost of gathering and processing the extra information.
- **Stability**
  - ✓ Fruitless migration of processes, known as processor thrashing, must be prevented.
  - ✓ E.g. if nodes n1 and n2 observe that node n3 is idle and then offload a portion of their work to n3 without being aware of the offloading decision made by the other node.
  - ✓ Now if n3 becomes overloaded due to this it may again start transferring its processes to other nodes.
  - ✓ This is caused by scheduling decisions being made at each node independently of decisions made by other nodes.

5.3.2 Task Assignment Approach

The following assumptions are made in this approach:

- A process has already been split up into pieces called **tasks.**
- This split occurs along natural boundaries (such as a method), so that each task will have integrity in itself and data transfers among the tasks are minimized.
- The amount of computation required by each task and the speed of each CPU are known. The cost of processing each task on every node is known. This is derived from assumption 2.
- The IPC cost between every pair of tasks is already known.
- The IPC cost is 0 for tasks assigned to the same node.

- This is usually estimated by an analysis of the static program.
- If two tasks communicate *n* times and the average time for each inter-task communication is *t*, them IPC costs for the two tasks is *n * t*.
- Precedence relationships among the tasks are known.
- Reassignment of tasks is not possible.

Goal of this method is to assign the tasks of a process to the nodes of a distributed system in such a manner as to achieve goals such as:

- ❖ Minimization of IPC costs
- ❖ Quick turnaround time for the complete process
- ❖ A high degree of parallelism
- ❖ Efficient utilization of system resources in general

- These goals often conflict. E.g., while minimizing IPC costs tends to assign all tasks of a process to a single node, efficient utilization of system resources tries to distribute the tasks evenly among the nodes.
- So also, quick turnaround time and a high degree of parallelism encourage parallel execution of the tasks, the precedence relationship among the tasks limits their parallel execution.
- In case of *m* tasks and *q* nodes, there are m^q possible assignments of tasks to nodes .
- In practice, however, the actual number of possible assignments of tasks to nodes may be less than m^q due to the restriction that certain tasks cannot be assigned to certain nodes due to their specific requirements.

5.3.3 Classification of Load Balancing Algorithms

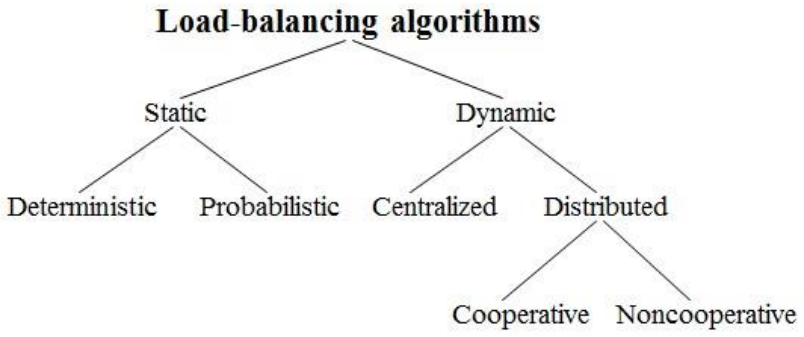


Fig 5.11: Classification of Load Balancing Algorithms

❖ **Static versus Dynamic**

Static Algorithms	**Dynamic Algorithms**
Static algorithms use only information about the average behavior of the system.	Dynamic algorithms collect state information and react to system state if it changed.
Static algorithms ignore the current state or load of the nodes in the system.	Dynamic algorithms are able to give significantly better performance.
Static algorithms are much simpler.	They are complex

❖ **Deterministic versus Probabilistic**

Deterministic Algorithms	**Probabilistic Algorithms**
Deterministic algorithms use the information about the properties of the nodes and the characteristic of processes to be scheduled.	Probabilistic algorithms use information of static attributes of the system (e.g. number of nodes, processing capability, topology) to formulate simple process placement rules
Deterministic approach is difficult to optimize.	Probabilistic approach has poor performance

❖ **Centralized versus Distributed**

Centralized Algorithms	**Distributed Algorithms**
Centralized approach collects information to server node and makes assignment decision.	Distributed approach contains entities to make decisions on a predefined set of nodes
Centralized algorithms can make efficient decisions, have lower fault-tolerance	Distributed algorithms avoid the bottleneck of collecting state information and react faster

❖ **Cooperative versus Non-cooperative**

Cooperative Algorithms	**Non-cooperative Algorithms**
In Co-operative algorithms distributed entities cooperatewith each other.	In Non-cooperative algorithms entities act as autonomous ones and make scheduling decisions independently from other entities.

Cooperative algorithms are more complex and involve larger overhead.	They are simpler.
Stability of Cooperative algorithms are better.	Stability is comparatively poor.

Issues in Load Balancing Algorithms

- **Load estimation policy**: determines how to estimate the workload of a node.
- **Process transfer policy:** determines whether to execute a process locally or remote.
- **State information exchange policy:** determines how to exchange load information among nodes.
- **Location policy:** determines to which node the transferable process should be sent.
- **Priority assignment policy:** determines the priority of execution of local and remote processes.
- **Migration limiting policy:** determines the total number of times a process can migrate.

Load Estimation Policies:

Policy I:

- To balance the workload on all the nodes of the system, it is necessary to decide how to measure the workload of a particular node.
- Some measurable parameters (with time and node dependent factor) can be the following:
  - Total number of processes on the node
  - Resource demands of these processes
  - Instruction mixes of these processes
  - Architecture and speed of the node's processor
- Several load-balancing algorithms use the total number of processes to achieve big efficiency.

Policy II:

- In some cases the true load could vary widely depending on the remaining service time, which can be measured in several way:
    - **Memory less** method assumes that all processes have the same expected remaining service time, independent of the time used so far.
    - **Past repeats** assumes that the remaining service time is equal to the time used so far.
    - **Distribution** method states that if the distribution service times is known, the associated process's remaining service time is the expected remaining time conditioned by the time already used.

Policy III:

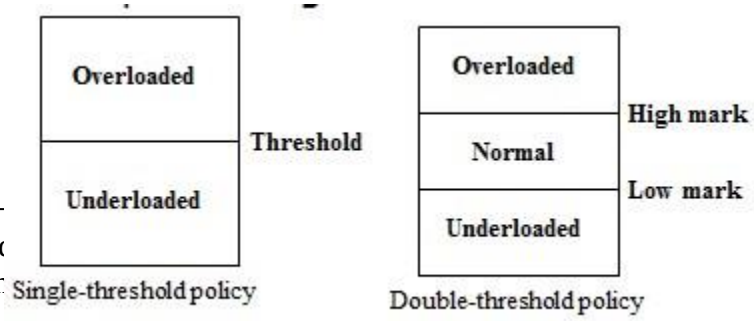
- None of the previous methods can be used in modern systems because of periodically running processes and daemons.
- An acceptable method for use as the load estimation policy in these systems would be to measure the CPU utilization of the nodes.
- Central Processing Unit utilization is defined as the number of CPU cycles actually executed per unit of real time.
- It can be measured by setting up a timer to periodically check the CPU state (idle/busy).

Threshold Policy

- Most of the algorithms use the **threshold policy** to decide on whether the node is lightly-loaded or heavily-loaded.
- Threshold value is a limiting value of the workload of node which can be determined by:
    - Static policy: predefined threshold value for each node depending on processing capability.
    - Dynamic policy: threshold value is calculated from average workload and a predefined constant
- Below threshold value node accepts processes to execute, above threshold value node tries to transfer processes to a lightly-loaded node.

Single threshold Policy

- Single-threshold policy may lead to unstable algorithm because under loaded node could turn to be overloaded right after a process migration.



- To rec ed which is also knowr

Fig 5.12: Process Threshold Policy

Double threshold Policy

- When node is in overloaded region new local processes are sent to run remotely, requests to accept remote processes are rejected.
- When node is in normal region new local processes run locally, requests to accept remote processes are rejected.
- When node is in under loaded region new local processes run locally, requests to accept remote processes are accepted

Location Policies:

- **Threshold method:**

  This policy selects a random node, checks whether the node is able to receive the process, then transfers the process. If node rejects, another node is selected randomly. This continues until probe limit is reached.

- **Shortest method**

  * L distinct nodes are chosen at random, each is polled to determine its load. The process is transferred to the node having the minimum value unless its workload value prohibits to accept the process.

  * Simple improvement is to discontinue probing whenever a node with zero load is encountered.

- **Bidding method**

  * The nodes contain managers (to send processes) and contractors (to receive processes).

- The managers broadcast a request for bid, contractors respond with bids (prices based on capacity of the contractor node) and manager selects the best offer.
- Winning contractor is notified and asked whether it accepts the process for execution or not.
- The full autonomy for the nodes regarding scheduling.
- This causes big communication overhead.
- It is difficult to decide a good pricing policy.

➢ **Pairing**

- Contrary to the former methods the pairing policy is to reduce the variance of load only between pairs.
- Each node asks some randomly chosen node to form a pair with it.
- If it receives a rejection it randomly selects another node and tries to pair again.
- Two nodes that differ greatly in load are temporarily paired with each other and migration starts.
- The pair is broken as soon as the migration is over.
- A node only tries to find a partner if it has at least two processes.

Priority assignment policy

❖ **Selfish:** Local processes are given higher priority than remote processes. Worst response time performance of the three policies.

❖ **Altruistic:** Remote processes are given higher priority than local processes. Best response time performance of the three policies.

❖ **Intermediate:** When the number of local processes is greater or equal to the number of remote processes, local processes are given higher priority than remote processes. Otherwise, remote processes are given higher priority than local processes.

Migration limiting policy

This policy determines the total number of times a process can migrate

➢ **Uncontrolled:** A remote process arriving at a node is treated just as a process originating at a node, so a process may be migrated any number of times

➢ **Controlled:**

- Avoids the instability of the uncontrolled policy.

- Use a migration count parameter to fix a limit on the number of time a process can migrate.
- Irrevocable migration policy: migration count is fixed to 1.
- For long execution processes migration count must be greater than 1 to adapt for dynamically changing states

5.3.4 Load-sharing approach

- The following problems in load balancing approach led to load sharing approach:
  - Load balancing technique with attempting equalizing the workload on all the nodes is not an appropriate object since big overhead is generated by gathering exact state information.
  - Load balancing is not achievable since number of processes in a node is always fluctuating and temporal unbalance among the nodes exists every moment

Basic idea:

- It is necessary and sufficient to prevent nodes from being idle while some other nodes have more than two processes.
- Load-sharing is much simpler than load-balancing since it only attempts to ensure that no node is idle when heavily node exists.
- Priority assignment policy and migration limiting policy are the same as that for the load-balancing algorithms.

Load Estimation Policies

- Since load-sharing algorithms simply attempt to avoid idle nodes, it is sufficient to know whether a node is busy or idle.
- Thus these algorithms normally employ the simplest load estimation policy of counting the total number of processes.
- In modern systems where permanent existence of several processes on an idle node is possible, algorithms measure CPU utilization to estimate the load of a node

Process Transfer Policies

- The load sharing algorithms normally use all-or-nothing strategy.
- This strategy uses the threshold value of all the nodes fixed to 1.
- Nodes become receiver node when it has no process, and become sender node when it has more than 1 process.

- ✓ To avoid processing power on nodes having zero process load-sharing algorithms uses a threshold value of 2 instead of 1.
- ✓ When CPU utilization is used as the load estimation policy, the double-threshold policy should be used as the process transfer policy

Location Policies

- ❖ The location policy decides whether the sender node or the receiver node of the process takes the initiative to search for suitable node in the system, and this policy can one of the following:
- ❖ **Sender-initiated location policy:** Sender node decides where to send the process. Heavily loaded nodes search for lightly loaded nodes
- ❖ **Receiver-initiated location policy**: Receiver node decides from where to get the process. Lightly loaded nodes search for heavily loaded nodes

Sender-initiated location policy

- ➢ Node becomes overloaded, it either broadcasts or randomly probes the other nodes one by one to find a node that is able to receive remote processes.
- ➢ When broadcasting, suitable node is known as soon as reply arrives

Receiver-initiated location policy

- ➢ Nodes becomes underloaded, it either broadcast or randomly probes the other nodes one by one to indicate its willingness to receive remote processes.
- ➢ Receiver-initiated policy require preemptive process migration facility since scheduling decisions are usually made at process departure epochs
  - ✓ Both policies gives substantial performance advantages over the situation in which no load-sharing is attempted.
  - ✓ Sender-initiated policy is preferable at light to moderate system loads.
  - ✓ Receiver-initiated policy is preferable at high system loads.
  - ✓ Sender-initiated policy provide better performance for the case when process transfer cost significantly more at receiver-initiated than at sender-initiated policy due to the pre-emptive transfer of processes.

State information exchange policies

- ∗ In load-sharing algorithms it is not necessary for the nodes to periodically exchange state information, but needs to know the state of other nodes when it is either underloaded or overloaded.

* The following are the two approaches followed when there is state change:

* **Broadcast when state changes**

  – In sender-initiated/receiver-initiated location policy a node broadcasts State Information Request when it becomes overloaded/ underloaded.

  – It is called broadcast-when-idle policy when receiver-initiated policy is used with fixed threshold value of 1

* **Poll when state changes**

  – In large networks polling mechanism is used.

  – Polling mechanism randomly asks different nodes for state information until find an appropriate one or probe limit is reached.

  – It is called poll-when-idle policy when receiver-initiated policy is used with fixed threshold value value of 1.

REVIEW QUESTIONS

PART - A

1. Define process management.

Process Management involves the activities aimed at defining a process, establishing responsibilities, evaluating process performance, identifying and opportunities or improvement.

2. Define process migration.

Process migration is the act of transferring process between two machines.

3. What is Eager (All)?

* This transfers the entire address space.
* No trace of process is left behind in the original system.
* If address space is large and if the process does not need most of it, then this approach is unnecessarily expensive.
* Implementations that provide a checkpoint/restart facility are likely to use this approach, because it is simpler to do the check-pointing and restarting if all of the address space is localized.

4. What is Precopy?

* In this method, the process continues to execute on the source node while the address space is copied to the target node.
* The pages modified on the source during the precopy operation have to be copied a second time.
* This strategy reduces the time that a process is frozen and cannot execute during migration.

5. What is Eager (dirty)?

* The transfer involves only the portion of the address space that is in main memory and has been modified.
* Any additional blocks of the virtual address space are transferred on demand.
* The source machine is involved throughout the life of the process.

6. What is Copy-on-reference?

* In this method, the pages are only brought over when referenced.
* This has the lowest initial cost of process migration.

7. What is Flushing?

* The pages are cleared from main memory by flushing dirty pages to disk.
* This relieves the source of holding any pages of the migrated process in main memory.

8. Define thread.

A thread is the entity within a process that can be scheduled for execution. All threads of a process share its virtual address space and system resources.

9. What is thread context?

The *thread context* includes the thread's set of machine registers, the kernel stack, a thread environment block, and a user stack in the address space of the thread's process.

10. What is thread pool?

Thread pool consists of a number of threads in a pool where they await work. This is slightly faster to service a request with an existing thread than create a new thread. This allows the number of threads in the application(s) to be bound to the size of the pool.

11. What is worker thread?

A Worker Thread has a persistent state, and will be invoked by start(). The life of worker thread is till the calling object goes out of scope, or is explicitly shutdown

12. What is job object?

A job object allows groups of processes to be managed as a unit.

13. What is UMS?

User-mode scheduling (UMS) is a lightweight mechanism that applications can use to schedule their own threads.

14. What is a fiber?

A fiber is a unit of execution that must be manually scheduled by the application.

15. What is multithreaded program?

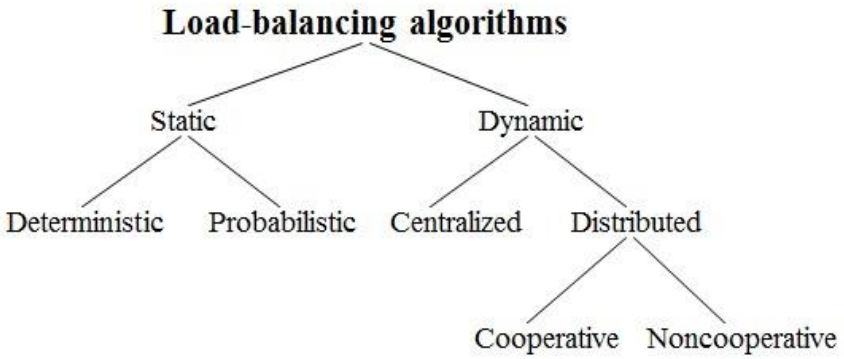
A multi- threaded program contains two or more parts that can run concurrently and each part can handle different task at the same time making optimal use of the available resources especially when your computer has multiple CPUs.

16. List the features of scheduling algorithm.

- General purpose
- Efficiency

- Fairness
- Dynamic
- Transparency
- Scalability
- Fault tolerance
- Quick decision making capability
- Balanced system performance and scheduling overhead
- Stability

17. Classify load balancing algorithms.



18. Differentiate static and dynamic algorithms.

Static Algorithms	**Dynamic Algorithms**
Static algorithms use only information about the average behavior of the system.	Dynamic algorithms collect state information and react to system state if it changed.
Static algorithms ignore the current state or load of the nodes in the system.	Dynamic algorithms are able to give significantly better performance.
Static algorithms are much simpler.	They are complex

19. Differentiate deterministic and probabilistic algorithms.

Deterministic Algorithms	**Probabilistic Algorithms**
Deterministic algorithms use the information about the properties of the nodes and the characteristic of processes to be scheduled.	Probabilistic algorithms use information of static attributes of the system (e.g. number of nodes, processing capability, topology) to formulate simple process placement rules
Deterministic approach is difficult to optimize.	Probabilistic approach has poor performance

20. Differentiate centralized and distributed algorithms.

Centralized Algorithms	**Distributed Algorithms**
Centralized approach collects information to server node and makes assignment decision.	Distributed approach contains entities to make decisions on a predefined set of nodes
Centralized algorithms can make efficient decisions, have lower fault-tolerance	Distributed algorithms avoid the bottleneck of collecting state information and react faster

21. Differentiate cooperative and non co operative algorithms.

Cooperative Algorithms	**Non-cooperative Algorithms**
In Co-operative algorithms distributed entities cooperatewith each other.	In Non-cooperative algorithms entities act as autonomous ones and make scheduling decisions independently from other entities.
Cooperative algorithms are more complex and involve larger overhead.	They are simpler.
Stability of Cooperative algorithms are better.	Stability is comparatively poor.

22. Give the issues in Load Balancing Algorithms

- ❖ Load estimation policy: determines how to estimate the workload of a node.
- ❖ Process transfer policy: determines whether to execute a process locally or remote.
- ❖ State information exchange policy: determines how to exchange load information among nodes.
- ❖ Location policy: determines to which node the transferable process should be sent.
- ❖ Priority assignment policy: determines the priority of execution of local and remote processes.
- ❖ Migration limiting policy: determines the total number of times a process can migrate.

23. What is threshold policy?

Most of the algorithms use the threshold policy to decide on whether the node is lightly-loaded or heavily-loaded. Threshold value is a limiting value of the workload of node which can be determined by:

- ∗ Static policy: predefined threshold value for each node depending on processing capability.
- ∗ Dynamic policy: threshold value is calculated from average workload and a predefined constant

24. What is single threshold policy?

Single-threshold policy may lead to unstable algorithm because under loaded node could turn to be overloaded right after a process migration.

25. What is double threshold policy?

When node is in overloaded region new local processes are sent to run remotely, requests to accept remote processes are rejected. When node is in normal region new local processes run locally, requests to accept remote processes are rejected.When node is in under loaded region new local processes run locally, requests to accept remote processes are accepted

26. What are the location policies?

- ➢ Threshold method
- ➢ Shortest method
- ➢ Bidding method
- ➢ Pairing
- ➢ Threshold method:

27. What is threshold policy?

This policy selects a random node, checks whether the node is able to receive the process, then transfers the process. If node rejects, another node is selected randomly. This continues until probe limit is reached.

28. What is Shortest method?

- L distinct nodes are chosen at random, each is polled to determine its load. The process is transferred to the node having the minimum value unless its workload value prohibits to accept the process.
- Simple improvement is to discontinue probing whenever a node with zero load is encountered.

29. What is Bidding method?

- The nodes contain managers (to send processes) and contractors (to receive processes).
- The managers broadcast a request for bid, contractors respond with bids (prices based on capacity of the contractor node) and manager selects the best offer.
- Winning contractor is notified and asked whether it accepts the process for execution or not.
- The full autonomy for the nodes regarding scheduling.
- This causes big communication overhead.
- It is difficult to decide a good pricing policy.

30. What is Pairing?

- Contrary to the former methods the pairing policy is to reduce the variance of load only between pairs.
- Each node asks some randomly chosen node to form a pair with it.
- If it receives a rejection it randomly selects another node and tries to pair again.
- Two nodes that differ greatly in load are temporarily paired with each other and migration starts.
- The pair is broken as soon as the migration is over.
- A node only tries to find a partner if it has at least two processes.

31. What is Priority assignment policy?

- ❖ Selfish: Local processes are given higher priority than remote processes. Worst response time performance of the three policies.
- ❖ Altruistic: Remote processes are given higher priority than local processes. Best response time performance of the three policies.
- ❖ Intermediate: When the number of local processes is greater or equal to the number of remote processes, local processes are given higher priority than remote processes. Otherwise, remote processes are given higher priority than local processes.

32. What are the issues in load balancing approach?

The following problems in load balancing approach led to load sharing approach:

- Load balancing technique with attempting equalizing the workload on all the nodes is not an appropriate object since big overhead is generated by gathering exact state information.
- Load balancing is not achievable since number of processes in a node is always fluctuating and temporal unbalance among the nodes exists every moment

33. What is load sharing approach?

It is necessary and sufficient to prevent nodes from being idle while some other nodes have more than two processes. Load-sharing is much simpler than load-balancing since it only attempts to ensure that no node is idle when heavily node exists. Priority assignment policy and migration limiting policy are the same as that for the load-balancing algorithms.

34. What are location policies?

- ❖ The location policy decides whether the sender node or the receiver node of the process takes the initiative to search for suitable node in the system, and this policy can one of the following:
- ❖ Sender-initiated location policy: Sender node decides where to send the process. Heavily loaded nodes search for lightly loaded nodes
- ❖ Receiver-initiated location policy: Receiver node decides from where to get the process. Lightly loaded nodes search for heavily loaded nodes

PART-B

1. Explain in detail about process migration.
2. Define threads. Explain all the multithreading thread models.
3. What are the issues in threads?
4. Discuss the various thread implementations.
5. Describe the resource management in distributed systems.
6. Write about task assignment.
7. Classify load balancing algorithms.
8. Explain load sharing approaches.