REVIEW QUESTIONS

1. What is cryptographic system? Give a general model of cryptographic system. What are public and private keys? (UPTU 2003)
2. Discuss the public key scheme in cryptography. Why is it assumed as suitable method for securing data in environment having multiple users. (UPTU 2004)
3. Show how secrete key cryptography can be used to establish secure communication channel in client server architecture. (UPTU 2005)
4. What is digital signature? How is it used in generating and verifying signatures ? (UPTU 2006, 07)
5. Writes short notes on :
   (a) RSA algorithm
   (b) Diffic-Hellman Algorithm
6. What is firewall? Explain various categories of firewall.
7. Differentiate public and secret key encryption. How they provide authentication and confidentiality?

□□□□

IV

Unit - IV

(1)

Chapter 14
Distributed File Systems

14.1. INTRODUCTION

A distributed file system is a resource management component of distributed operating system. It implements a common file system that can be shared by all the autonomous computers in the system. It supports the sharing of information in the form of files throughout intranet. A well designed file service provides access to file stored at a server with performance and reliability similar to files stored on local disks. A distributed file system enables programs to store and access remote files and allows users to access files from any computer in an intranet.

Two important goals of distributed file systems are :

(i) Network Transparency : The primary goal is to provide the same functional capabilities to access files distributed over a network as the file system of a time sharing mainframe system does to access files residing at one location. This property is called network transparency.

(ii) High Availability : Another major goal is to provide high availability. Users should have the same way, the same easy access to files, irrespective of their physical locations. System failure or regular scheduled activities such as backups or maintenance should not result in unavailability of files.

14.2. ARCHITECTURE OF DISTRIBUTED FILE SYSTEM

Ideally in distributed file system, file can be stored at any machine (or computer) and the computation can be performed at any machine. When a machine needs to access a file stored on a remote machine, the remote machine performs the necessary file access operations and returns data if a read operation is performed.

For higher performance, several machines referred to as file servers are dedicated to storing files and performing storage and retrieval operations. The rest of machine in the systems can be used solely for computational purpose. These machines are referred to as clients and they access the files stored on servers.

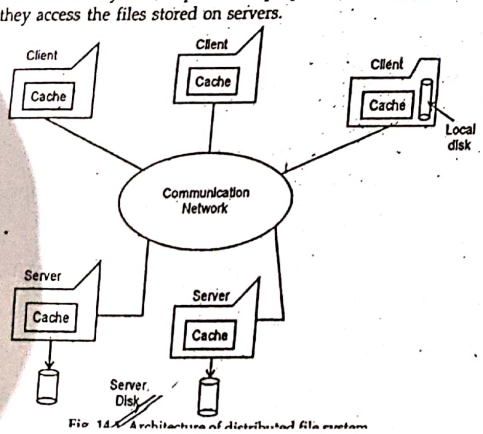


Fig. 14.1. Architecture of distributed file system

Services present in distributed file system are :

1. **Name Server :** It is a process that maps names specified by clients to such files and directories. The maping (name resolution) occurs when references a file or directory for the first time.
2. **Cache Manager :** It is a process that implements file caching. In file caching of data stored at a remote file server is brought to the client's machine referenced by client. Subsequent access to the data are performed locally client, thereby reducing the access delays due to network lantency. Cache mana can be present both on servers and clients.

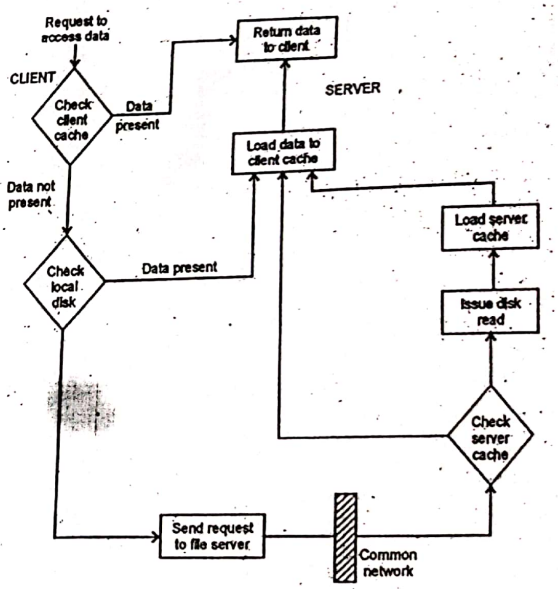


Fig. 14.2. Data access actions in distributed file system.

A request by a process to access a data block is present to the local cache (client cache) of machine (client), on which the process is running. If the block is not in the cache, then check local disk. If present, then check for the presence of data block. If the block is present then the request is satisfied and the block is loaded in to the client cache. If the block is not stored locally, then the request is passed on to the appropriate file server. The server checks its own cache for the presence of data block before issuing a disk I/O request. The data block is finally transferred to the client cache and loaded to the server cache if it was missing in server cache.

...ism for Building Distributed File Systems

...here are five mechanisms :

1. Mounting
2. Caching
3. Hints
4. Bulk Data Transfer
5. Encryption

14.2.1. Mounting

A mount mechanism allows the binding together of different file name spaces to form a single hierarchically structured name space. Mounting is UNIX specific and most of the existing distributed file systems are UNIX based.

A name space (or collection of files) can be bounded to or mounted at an internal node or a leaf node of a name space tree.

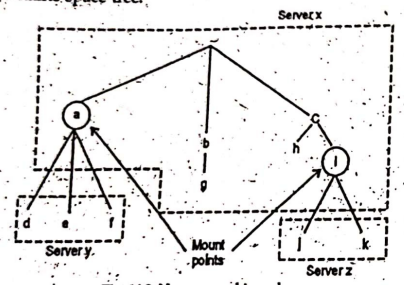


Fig. 14.3. Name-space hierarchy.

A node, onto which a name space is mounted is known as mount point. In Fig. 14.3. *a* and *i* are mount points at which directories stored at server *y* and server *z* are mounted respectively. Node *a* and *i* are internal nodes in the name space tree. The kernel maintains a structure called the mount table, which maps mount points to appropriate storage devices.

In case of distributed file system, file system maintained by remote servers are mounted at the clients. There are two approaches to maintain mount informations :

(i) Mount information can be maintained at client, in which each client has to individually mount every required file system. This approach is employed in Sun network file system.

(ii) Mount information can be maintained at servers, in which it is possible that every client sees an identical file name space. *y* files are moved to different server, then mount information need only be updated at servers.

14.2.2. Caching

Caching is commonly employed in distributed file systems to reduce delays in the accessing of data. In file caching, a copy of data stored at a remote file server is brought to the client when referenced by client, thereby reducing access delays due to network latency. Caching exploits the temporal locality of reference exhibited by programs. It refers to the fact that a file recently accessed is likely to be accessed again in near future. Data can either be cached in main

memory (server cache) at the servers to reduce disk access latency. In addition, caching reduces the frequency of access of the file servers and the communication network, thereby improving the scalability of file system. The file system performance can be improved by caching since accessing remote disks is much slower than accessing local memory or local disk.

14.2.3. Hints

Caching improves file system performance by reducing the delay in accessing data. However, when multiple clients cache and modify shared data, the problem of cache consistency arises. Guaranteeing consistency in DFS is expensive as it requires elaborate cooperation between file servers and clients.

An alternative approach is to treat cached data as hints. In this case, cached data are not expected to be completely accurate, however, valid cache entries improve performance substantially without incuring the cost.

The class of application that can utilize hints are those, which can recover after discovering that the cached data are invalid. For example after a name of file or directory is mapped to object, the address of the object can be stored as hint in the cache.

14.2.4. Bulk Data Transfer

In bulk data transfer, multiple consecutive data blocks are transferred from servers to clients instead of just the block reference by clients. Bulk transfer amortizes the portocol processing overhead and disk seek time over many consecutive blocks of a file. Bulk transfer reduces file access overhead by obtaining a multiple number of block in single seek.

14.2.5. Encryption

It is used for enforcing security in DS. Needham-Schroeder protocol is the basis of security mechanism in distributed system. Here conversation key is transferred in encrypted form as cipher.

14.3. FILE SERVICE ARCHITECTURE

The scope for open, configurable system is enchanced if the file service is structured as three components — a *flat file service*, a *directory service* and a *client module*. The relevant modules and their relationships are shown in Fig. 14.4. The flat file service and the directory service each export an interface for use by client programs, and their RPC interfaces, taken together, provide a comprehensive set of operations for access to file. The client module provides a single programming interface with operations on files similar to those found in conventional file system. The design is *open* in the sense that different client modules can be used to implement different programming interfaces, simulating the file operations of different operating systems and optimizing the performance for different clients and server hardware configurations.

The division of responsibilities between the modules can be defined as follows :

1. Flat File Service : The flat file service is concerned with implementing operations on the contents of files. *Unique file identifiers* (UFIDs) are used to refer to files in all requests for flat file service operation. The division of responsibilities between the file service and the directory service is based upon the use of UFIDs. UFIDs are long sequence of bits chosen so that each file has a UFID that a unique among all of the files in a distributed system. When the flat file service receives a request to create a file, it generates a new UFID for it and returns the UFID to the requester.

2. **Directory Service :** The directory service provides a mapping between *text names* for files and their UFIDs. Clients may obtain the UFID of a file by quoting its text names to the directory service. The directory service provides the functions needed to generate directories to add new file names to directories and to obtain UFIDs from directories. It is a client of the flat file service; its directory files are stored in files of the flat file service. When a hierarchic file-naming scheme is adopted, as in UNIX, directories hold references to other directories.

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

Flat File Service Operations :

Read(filed, i, n) → *Data* – throws *BadPosition*	If $1 \le i \le length$ *(file)* :Reads a sequence of upto *n* items for a file starting at time *i* and returns it in *Data*.
Write(Field, i, Data) – throws *BadPosition*	If $1 \le i \le length$ *(file) + 1* : Writes a sequence of *Data* to a file, starting at item *i*, extending the file, if necessary.
Creat() → *Field*	Creates a new file of length 0 and delivers a UFID for it.
Delete (Field)	Removes the file from the file store.
GetAttributes(Field) → *Attr*	Returns the file attributes for the file.
SetAttributes(Field, Attr)	Sets the file attributes (only those attributes that are not shaded in Fig. 14.4).

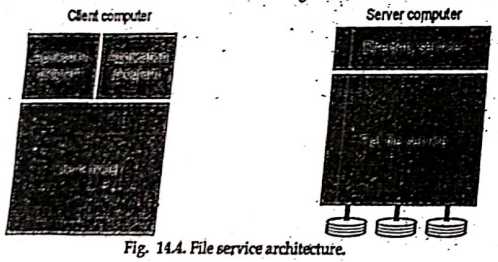


Fig. 14.4. File service architecture.

Flat File Service Interface

Fig. 14.5 contains a definition of the interface to a flat file service. This is the RPC interface used by client moudles. It is not normally used directly by user-level programs. A *Field* is invalid if the file that it refers to is not present in the server processing the request or if its access permissions are inappropriate for the operation requested. All of the procedures in the interface except *Create* throw exceptions, if the *Field* argument contains an invalid UFID or the user does not have sufficient access rights. These exceptions are omitted from the definition for clarity.

The most important operations are those for reading and writing. Both the *Read* and the *Write* operation require a parameter *i* specifying a position in the file. The *Read* operation copies the sequence of *n* data items beginning at item *i* from the specified file in to *Data*, which is then returned to the client. The *Write* operation copies the sequence of data items in *Data* into the specified file beginning at item *i*, replacing the previous contents of file at the corresponding position and extending the file, if necessary.

Create creates a new, empty file and returns the UFID that is generated. It is not specified file.

GetAttributes and *SetAttributes* enable clients to access the attributes record. Get is normally available to any client that is allowed to read the file. Access to the Set operation would normally be restricted to the directory service that provides access to The values of the length and timestamp portions of the attributes record are not affected *SetAttributes*; that are maintained separately by the flat file service itself.

Directory Service Interface

Fig. 14.5 contains a definition of the RPC interface to a directory service. The purpose of the directory service is to provide a service for translating text names to UFIDs. In order to do so, it maintains directory files containing the mappings between text names for file and UFIDs. Each directory is stored as a conventional file with a UFID, so the directory service is a client of the file service.

There are three operations for alternating directories : *AddName Rename* and *UnName*. *AddName* adds an entry to a directory and increments the reference count filed in the file attribute record.

Directory Service Operations

Lookup(Dir, Name) → *Field* – throws *NotFound*	Locates the text name in the directory and returns the relevant UFID. If *Name* is not in the directory, throws an exception.
AddName(Dir, Name, File) – throws *Name Duplicate*	If *Name* is not in the directory, adds (*Name, File*) to the directory and updates the file's attribute record. If *Name* is already in the directory : throws an exception.
UnName(Dir, Name) – throws *NotFound*	If *Name* is in the directory : the entry containing *Name* is removed from the directory. If *Names* is not in the directory : throws an exception
GetNames(Dir, Pattern) → *Name seq.*	Returns all the text name in the directory that match the regular expression *Pattern*.

Hierarchic File System

A hierarchic file system such as the one that UNIX provides consists of a number of directories arranged in a tree structure. Each directory holds the names of the files and other directories that are accessible from it. Any file or directory can be referenced using a *pathname* – a multi-part name that represents a path through the tree. The root has a distinguished name, and each file or directory has a name in a directory. The UNIX file-naming scheme is not a strict hierarchy-files can have several names, and they can be in the same or different directories. This is implemented by a *link* operation, which adds a new name for a file to a specified directory.

File Grouping

A *file group* is a collection of file located on a given server. A server may hold several file groups, and groups can be moved between servers, but a file cannot change the group.

...longs. A similar construct (called a *filesystem*) is used in UNIX and in most other ...systems. File groups were originally introduced to support facilities for moving ...of files stored on removable disk cartridges between computers.

...group identifiers must be unique throughout a distributed system. Since file groups ...moved, and distributed systems that are initially separate can be merged to form a ...system, the only way to ensure that file group identifiers will always be distinct in a ...system is to generate them with an algorithm that ensures global uniqueness. For example, ...whenever a new file group is created, a unique identifier can be generated by concatenating ...the 32-bit IP address of the host creating the new group with a 16-bit integer derived from the ...data, producing a unique 48-bit integer :

14.4. SUN NETWORK FILE SYSTEMS (NFS)

The Sun Network File System (NFS™) provides transparent, remote access to file systems. Unlike many other remote file system implementations under UNIX ®, NFS is designed to be easily portable to other operating systems and machine architectures. It uses an External Data Representation (XDR) specification to describe protocol in a machine and system independent way. [illegible] Remote Procedure Call package.

Fig. 14.5. shows the architecture of Sun NFS. It follows the abstract model defined in the preceding section. All implementations of NFS support the NFS protocol — a set of remote procedure calls that provide the means for clients to perform operations on a remote file store. The NFS protocol is operating-system-independent but was originally developed for use in networks of UNIX systems. We shall described the UNIX implementation the NFS protocol (version 3).

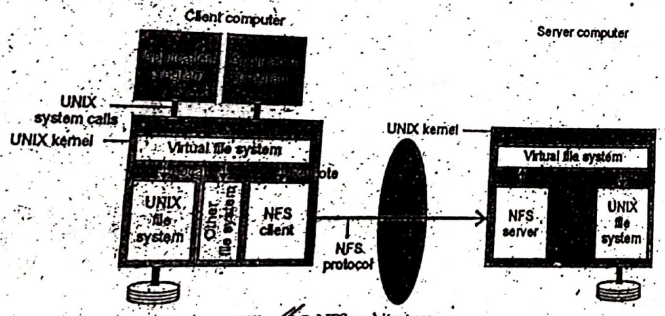


Fig. 14.5. NFS architecture.

14.1. Design Goals

The overall design goals of NFS were :

1. **Machine and Operating System Independence :** The protocols used should be independent of UNIX so that an NFS server can supply files to different types of clients. The protocols should also be simple enough that they can be implemented on low-end machines like the PC.
2. **Crash Recovery :** When clients can mount remote file system from different servers it is very important that clients and servers be able to recover easily from machine crashes and network problems.
3. **Transparent Access :** We want to provide a system, which allows programs to access remote file in exactly the same way as local files, without special, pathname parsing libraries, or recompiling. Programs should not need or be able to tell whether a file is remote or local.
4. **UNIX Semantics Maintained on UNIX Clients :** In order for transparent access to work on UNIX machines, UNIX file system semantics have to be maintained for remote files.
5. **Reasonable Performance :** People will not use a remote file system if it is no faster than the existing networking utilities, such as *rep*, even if it is easier to use. Our design goal was to make NFS as fast as a small local disk on a SSI interface.

Basic Design

The NFS design consists of three major pieces : the protocol, the server side and the client side.

NFS Protocol

The NFS protocol uses the Sun Remote Procedure Call (RPC) mechanism. For the same reasons that procedure calls simplify programs, RPC helps simplify the definition organization and implementation of remote services. The NFS protocol is defined in terms of a set of procedures, their arguments and results, and their effects. Remote procedure calls are synchronous, that is, the client application blocks until the server has completed the call and returned the results. This makes RPC very easy to use and understand because it behaves like a local procedure call.

NFS uses a stateless protocol. The parameters to each procedure call contain all information necessary to complete the call, and the server does not keep track of any past requests. This makes crash recovery very easy; when a server crashes, the client resends NFS requests until a response is received, and the server does not crash recovery at all. When a client crashes no recovery is necessary for either the client or the server.

An outline of the NFS protocol procedures is given below. For the complete specification see the *Sun Network File system Protocol Specification*.

null() returns () : Do nothing procedure to ping the server and measure round trip time.

lookup(dirfh, name) returns(fh, attr) : Returns a new fhandle and attributes for the named file in a directory.

create(dirfh, name, attr) returns (newfh, attr) : Creates a new file and returns its fhandle and attributes.

remove(dirfh, name) returns (status) : Removes a file from a directory.

getattr(fh) returns (attr) : Returns file attributes. This procedure is like a stat call.

Setattr(fh, attr) returns (attr) : Sets the mode, uid, gid, size, access time and modify time of a file. Setting the size to zero truncates the file.

Server Side

Because the NFS server is stateless, when servicing an NFS request it must commit any modified data to stable storage before returning results. The implication for UNIX based servers is the request that modify the file system that must flush all modified data to disk before returning from the call. For example, on a write request, not only the data block, but also any modified indirect blocks and the block containing the node must be flushed if they have been modified.

Client Side

The Sun implementation of the client side provides an interface to NFS, which is transparent to applications. To make transparent access to remote files work, we had to use a method of locating remote files that does not change the structure of path names. Some UNIX based remote file access methods use pathnames like *host:path* or */../host/path* to name remote files. This does not allow real transparent access since existing programs that parse pathnames have to be modified.

14.5. ANDREW FILE SYSTEM

AFS differs markedly from NFS in its desing and implementation. The differences are primarily attributable to the identification of scalability as the most important design goal. AFS is designed to perform well with larger numbers of active users than other distributed file systems. The key strategy for achieveing scalability is the caching of whole files in client nodes. AFS has two unusual desing characteristics :

Whole-file serving : The entire contents of directories and files are transmitted to client computers by AFS servers (in AFS-3, files larger than 64 kbytes are transferred in 64-kbyte chunks).

Whole-file caching : Once a copy of a file or a chunk has been transferred to a client computer it is stored in a cache on the local disk. The cache contains serveral hundred of the files most recently used on that computer. The cache is permanent.

Surviving reboots of the client computer. Local copies of files are used to satisfy clients' *open* request in preference to remote copies whenever possible.

Scenario

Here is a simple scenario illustrating the operation of AFS :

When a user process in a client computer issues an *open* system call for a file in the shared file space and there is no current copy of the file in the local cache then the server holding the file is located and a request for a copy of the file is sent.

The copy is stored in the local UNIX file system in the client computer, the copy is then *opened* and the resulting UNIX file descriptor is returned to the client.

Subsequent *read*, *write* and other operations on the file by processes in the client computer are applied to the local copy.

When the process in the client issues a *close* system call and if the local copy has been updated then its contents are sent back to the server. The server updates the file contents and the timestamp on the file. The copy on the client's local disk is retained in case it is needed again by a user-level process on the same workstation.

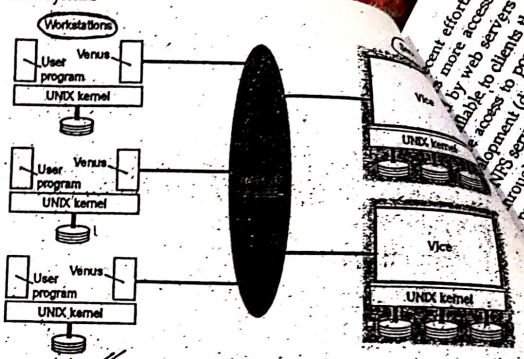


Fig. 14.6. Distribution of processes in the Andrew file system.

Most important observations are :

- Files are small : most are less than 10 kilobytes in size.
- Read operations on files are much more common than writes (about six times more common).
- Sequential access is common and random access is rater.
- Most files are read and written by only one user. When a file is shared, it is usually only one user who modifies it.
- Files are referenced in bursts. If a file has been referenced recently, there is a high probability that it will referenced again in the near future.

• AFS works best with the classes of file identified in the first position above. There is one important type of file that does not fit in to any of these classes – databases are typically shared by many users and are often updated quite frequently. The designers of AFS have explicitly excluded the provision of storage facilities for databases from their design goals, stating that the constraints imposed by different naming structures (that is, content-based access) and the need for fine-grained data access, concurrency control and atomicity of updates make it difficult to desing a distributed database system that is also a distributed file system.

14.6. RECENT ADVANCED

14.6.1. NFS Enhancements

Several research projects have addressed the question of one-copy update semantics by extending the NFS protocol to include *open* and *close* operations and adding a callback mechanism to enable the server to notify clients of the need to invalidate cache entries. We describe two such efforts here; their results seem to indicate that these enhancements can be accommodated without undue complexity or extra communication costs.

...cent efforts the Sun and other NFS developers have been directed at making ... more accessible and useful in widearea networks. While the HTTP protocol ... by web servers offers an effective and highly scalable method for making whole ...ilable to clients throughout the Internet, it is less useful to application programs that ...e access to portions of large files or those that update portions of files. The WebNFS ...elopment (described below) makes it possible for application programs to become clients ... NFS servers anywhere in the Internet (using the NFS protocol directly instead of indirectly through a kernel module). This, together with appropriate libraries for Java and other network programming languages, should offer the possibility of implementing Internet applications that share data directly, such as multi-user games or clients of large dynamic databases.

14.6.2. AFS Enhancements

We have mentioned that DCE/DFS, the distributed file system included in the Open Software Foundation's Distributed Computing Environment.

In AFS, callbacks are generated only when the server receives a *close* operation for a file that has been updated. DFS adopted a similar strategy to Spritely NFS and NQNFS to generate callbacks as soon as a file is updated. In order to update a file, a client must obtain a *write*.

14.7. DISTRIBUTED FILE SYSTEM REQUIREMENTS

The various distributed file system requirements are mentioned below :

Transparency

The file service is usually the most heavily loaded service is an intranet. So its functionality and performance are critical. The design of the file service should support many of the transparency requirements for distributed system identified.

The following forms of transparency are partially or wholly addressed by current file services :

Access transparency : Client programs should be unaware of the distribution of files. A single set of operations is provided for access to local and remote files. Programs written to operate on local files are able to access remote file without modification.

Local transparency : Client programs should see a uniform file name space. Files or groups of files may be relocated without changing their pathnames, and user programs see the same name space wherever they are executed.

Mobility transparency : Neither client programs nor system administration tables in client nodes need to be changed when files are moved. This allow file mobility— file or, more commonly, sets or volumes of files may be moved, either by system administrators or automatically.

Performance transparency : Client programs should continue to perform satisfactorily while the load on the service varies within a specified range.

Scaling transparency : The service can be expanded by incremental growth to deal with a wide range of load and network sizes.

Concurrent File Updates

Changes to a file by one client should not interfere with the operation of other clients simultaneously accessing or changing the same file. This is the well-known issue of concurrency control, discussed in detail in Chapter 15. The need for concurrency control for access to shared data in many applications is widely accepted and techniques are known for its implementation, but they are costly.

File Replication

In a file service that supports replication, a file may be represented by several copies of its contents at different locations. This has two benefits—it enables multiple servers to share the load of providing a service to clients accessing the same set of files, enhancing the scalability of the service, and it enhances faults tolerance by enabling clients to locate another service that holds a copy of the file when one has failed.

Hardware and Operating System Heterogeneity

The service interfaces should be defined so that client and server software can be implemented for different operating systems and computers. This requirement is an important aspect of openness.

Fault Tolerance

The central role of the file service in distributed system makes it essential that the service continue to operate in the face of client and server failures. Fortunately, a moderately fault-tolerant design is straight forward for simple servers.

Consistency

Conventional file systems such as that provided in UNIX offer *one-copy update semantic*. This refers to a model for concurrent access to file, in which the files contents seen by all of the processes accessing or updating a given file, in which the file are those that they would see if only a single copy of the file content existed. When files are replicated or cached at different sites, there is an inevitable delay in the propagation of modifications.

Security

Virtually all file systems provide access control mechanism based on the use of access control lists. In distributed files systems, there is a need to authenticate client request so that access control at the server is based on correct user identities and to protect the content of request and reply messages with digital signatures.

Efficiency

A distributed file service should offer facilities that are of at least the same power and generality as those found in conventional file systems and should achieve a comparable level of performance.

SOLVED EXAMPLES

1. Why is there no open or close operation in the interface to the flat service or the directory service ? What are the differences between our directory service Lookup operation and the UNIX open?

Ans. Because both services are stateless, the interface to the flat file service is designed to make *open* unnecessary. The *Lookup* operation performs a single-level lookup returning the UFID corresponding to a given simple name in a specified director To look up a pathname, a sequence of *Lookups* must be used. Unix *open* takes pathname and returns a file descriptor for the named file or directory.

2. Explain why the RPC interface to early implementations of NPS is potentia insecure.The security loophole has been closed in NFS 3 by the use of encrypt How is the encryption key kept secret ? Is the security of the key adequate ?

Ans. The user id for the client process was passed in the RPCs to the server in unencrypted form. Any program could simulate the NFS client module and transmit RPC calls to a NFS server with the user id of any user, thus gaining unauthorized access to their files. DES encryption is used in NFS version 3. The encryption key is established at *mount* time. The mount protocol is, therefore, a potential target for a security attack. Any workstation could simulate the mount protocol, and once a target file system has been mounted, it could impersonate any user using the encryption agreed at mount time.

3. How does the NFS Automounter help to improve the performance and scalability of NFS?

Ans. The NFS *mount service* operates at system boot time or user login time at each workstation, mounting file system wholesale in case they will be using during the login session. This was found too cumbersome for some applications and produces large number of unused entries in mount tables. With the Automounter file systems need not be mounted until they are accessed. This reduces the size of mount tables (and hence the time to search them). A simple form of file system replication for read-only file systems can be achieved with the Automounter, enabling the load of accesses to frequently-used systems files to be shared between several NFS servers.

4. Which features of the AFS design make it more scalable than NFS? What are the limits on its scalability, assuming that servers can be added as required? Which recent developments offer greater scalability? *(UPTU 2007, 08)*

Ans. The load of RPC calls on AFS servers is much less than NFS servers for the same client workload. This is achieved by the elimination of all remote calls except those associated with *open* and *close* operations, and the use of the *callback* mechanism to maintain the consistency of clients caches (compared to the use of *getattributes* calls by the clients in NFS). The scalability of AFS is limited by the performance of the single server that holds the most-frequently accessed file volume (e.g., the volume containing */etc/passwd/etc/host*, or some similar system file). Since read-write files cannot be replicated in AFS, there is no way to distribute the load of access to frequently-used file.

Designs such as xFS and Frangipani offers greater scalability by separating the management and metadata operations from the data handling, and they reduce network traffic by locating files based on usage patterns.

REVIEW QUESTIONS

1. What is distributed file system? What are the different mechanisms for building distributed file system? Explain mounting and caching. *(UPTU-2003, 04)*
2. Differentiate AFS and NFS. What are its limitations and advantages?
3. Which features of AFS design make it more scalable than NFS? What are the limits on its scalability assuming that servers can be added as required? *(UPTU-2007, 08)*
4. Briefly describe the steps involved in the working Andrew file system.
5. What is file service architecture? What are various file service operations?
6. Explain various design goals of Sun Network File System (NFS).
7. Explain various distributed file system requirements.

Chapter
Distributed Algorithms

INTRODUCTION TO COMMUNICATION PROTOCOLS

In the field of telecommunications, a communications protocol is the set of standard rules for data representation, signaling, authentication and error detection required to send information over a communication channel. An example of a simple communications protocol adapted to voice communication is the case of a radio dispatcher talking to mobile stations. Communication protocols for digital computer network communication have features intended to ensure reliable interchange of data over an imperfect communication channel. Communication protocol is basically following certain rules so that the system works properly.

Here you will learn about computer network protocols, TCP/IP introduction, FTP, HTTP overview, X.25, SMTP and SNMP. The word protocol is derived from the Greek word "protocollon" which means a leaf of paper glued to manuscript volume. In computer protocols means a set of rules a communication language or set of standards between two or more computing devices. Protocols exist at the several levels of the OSI (open system interconnectivity) layers model.

In the telecommunication system, there are one more protocol at each layer of the telephone exchange. On the internet, there is a suite of the protocols known as TCP/IP protocols that are consisting of transmission control protocol, internet protocol, file transfer protocol, dynamic host configuration protocol, Border gateway protocol and a number of other protocols.

In the telecommunication, a protocol is set of rules for data representation, authentication, and error detection. The communication protocols in the computer networking are intended for the secure, fast and error free data delivery between two communication devices. Communication protocols follow certain rules for the transmission of the data.

Protocols Properties

Different protocols perform different functions so it is difficult to generalize the properties of the protocols. There are some basic properties of most of the protocols.

- Detection of the physical (wired or wireless connection)
- Handshaking
- How to format a message.
- How to send and receive a message.
- Negotiation of the various connections.
- Correction of the corrupted or improperly formatted messages.
- Termination of the session.

The widespread use of the communication protocols is a prerequisite to the internet. The term TCP/IP refers to the protocols suite and a pair of the TCP and IP protocols are the most important internet communication protocols. Most protocols in communication are layered together where the various tasks listed above are divided. Protocols stacks refer to the combination of the different protocols. The OSI reference model is the conceptual model that is used to represent the stacks. There are different network protocols that perform different functions. Following is the description of the some of the most commonly used protocols.

HTTP (Hyper Text Transfer Protocol)

Hyper text transfer protocol is a method of transmitting the information on the web. HTTP basically publishes and retrieves the HTTP pages on the World Wide Web. HTTP is a language that is used to communicate between the browser and web server. The information that is transferred using HTTP can be plain text, audio, video, images, and hyper text. HTTP is a request/response protocol between the client and server. Many proxies, tunnels, and gateways can be existing between the web browser (client) and server (web server). An HTTP client initializes a request by establishing a TCP connection to a particular port on the remote host (typically 80 or 8080). An HTTP server listens to that port and receives a request message from the client. Upon receiving the request, server sends back 200 OK messages, its own message, an error message or other message.

POP3 (Post Office Protocol)

In computing, e-mail clients such as (MS outlook, outlook express and thunderbird) use Post Office Protocol to retreive e mails from the remote server over the TCP/IP connection. Nearly all the users of the Internet service providers use POP 3 in the e mail clients to retrieve the emails from the e mail servers. Most email applications use POP protocol.

SMTP (Simple Mail Transfer Protocol)

Simple Mail Transfer Protocol is a protocol that is used to send the e mail messages between the servers. Most e mail systems and e mail clients use the SMTP protocol to send messages to one server to another. In configuring an e mail application, you need to configure POP, SMTP and IMPA protools in your e mail software. SMTP is a simple, text based protocol and one or more recipient of the message is specified and then the message is transferred. SMTP connection is easily tested by the Telnet utility. SMTP uses the by default TCP port number 25.

FTP (File Transfer Protocol)

FTP or file transfer protocol is used to transfer (upload/download) data from one computer to another over the internet or through or computer network. FTP is a most commonly communication protocol for transferring the files over the internet. Typically, there are two computers are involved in the transferring the files a server and a client The client computer that is running FTP client software such as Cuteftp and AceFTP etc., initiates a connection with the remote computer (server). After successfully connected with the server, the client computer can perform a number of the operations like downloading the files, uploading, renaming and deleting the files, creating the new folder etc. Virtually operating system supports FTP protocols.

IP (Internet Protocol)

An Internet protocol (IP) is a unique address or identifier of each computer or communciation devices on the network and internet. Any participating computer networking device such as routers, computers, printers, internet fax machines and switches may have their own unique IP address. Personal information about someone can be found by the IP address. Every domain on the internet must have a unique or shared IP address.

DHCP (Dynamic Host Configuration Protocols)

The DHCP or Dynamic Host Configuration Protocol is a set of rules used by a communication device such as router, computer or network adapter to allow the device to request and obtain and IP address from a server which has a list of the larger number of addresses. DHCP is a protocol that is used by the network computers to obtain the IP addresses and other settings such as gateway, DNS, subnet mask from the DHCP server. DHCP ensures that all the IP addresses are unique and the IP address management is done by the server and not by the human. The assignment of the IP addresses is expires after the predetermined period of time. DHCP works in four phases known as DORA such as Discover, Observer, Request and Authorize.

IMAP (Internet Message Access Protocol)

The Internet Message Access Protocol known as IMAP is an application layer protocol that is used to access to access the emails on the remote server. POP3 and IMAP are the two most commonly used email retrieval protocols. Most of the email clients such as outlook express, thunderbird and MS outlooks support POP3 and IMAP. The email message are generally stored on the email server and the users generally retreive these messages whether by the web browser or email clients. IMAP is generally used in the larger networks. IMAP allows users to access their messages instantly on their systems.

Arcnet

ARCNET is a local area network technology that uses token bus scheme for managing line sharing among the workstations. When a device on a network wants to send a message, it inserts a token that is set to 1 and when a destination device reads the message it resets the token to 0 so that the frame can be used by another device.

FDDI

Fiber distributed data interface (FDDI) provides a standard for data transmission in a local area network that can extend a range of 200 kilometers. The FDDI uses token ring protocol as its basis. FDDI local area network can support a large number of users and can cover a large geographical area. FDDI uses fiber optic as a standard communciation medium. FDDI uses dual attached token ring topolgy. A FDDI network contains two token rings and the primary ring offers the capacity of 100 Mbits/s. FDDI is an ANSI standard network and it can support 500 stations in 2 kilometers.

UDP

The user datagram protocol is a most important protocol of the TCP/IP suite and is used to send the short messages known as datagram. Common network applications that uses UDP and DNS, online games, IPTV, TFTP and VOIP. UDP is very fast and light weight. UDP is an unreliable connectionless protocol that operates on the transport layer and it is sometimes called Universal Datagram Protocol.

X.25

X.25 is a standard protocol suite for wide area networks using a phone line or ISDN system. The X.25 standard was approved by CCITT and ITU in 1976.

TFTP

Trivial File Transfer Protocol (TFTP) is a very simple file transfer pro... very basic features of the FTP. TFTP can be implemented in a very small amou... TFTP is useful for booting computers such as routers. TFTP is also used to tra... over the network. TFPT uses UDP and provides no security features.

SNMP

The simple network management protocol (SNMP) forms the TCP/IP suite. S... used to manage the network attached devices of the complex network.

PPTP

The point to point tunneling protocol is used in the virtual private networks. PPP wo... by sending regular PPP session. PPTP is a method of implementing VPN networks.

15.1.1. OSI Reference Model

OSI reference model is now considered as a primary standard for internet working and inter computing. Today many network communication protocols are based on the standards of OSI model. In the OSI model the network/data communication is defined into seven layers. These 7 layers further divide the tasks of moving the data across the network into subtask and hence complete one communication cycle between two computers or two network devices. Each layer is assigned a task and the task is completed independently. The OSI layers have the clear and independent characteristics and tasks.

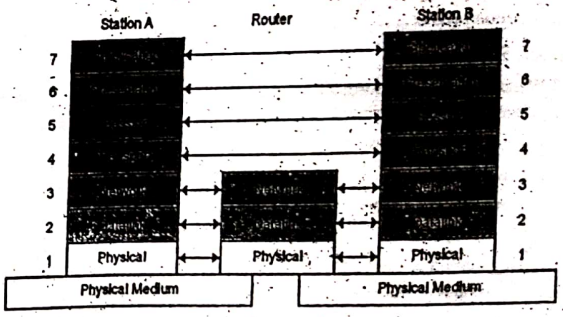


Fig. 15.1. OSI reference Model

The 7 layers of the OSI models can be divided into upper and lower layers. It have been defined the characteristcis, tasks and features of each layer separately.

Layer 7 : Application Layer

The application layer defines the interfaces for communication and data transfer. This layer also provides and support services such as job transfer, handles network access, e-mail, supports user applications and error recovery.

...ols : FTP, DNS, SNMP, SMTP, FINGER, TELENET, TFTP, BOOTP and SMB ...nd operated on the application layer.

...twork Devices : Gateway network device is operated on the application layer.

...yer 6 : Presentation Layer

The presentation layer present the data into a uniform format and masks the difference of data format between two dissimilar systems. It also translates the data from application to the network format. Presentation layer is also responsible for the protocol conversion, encryption, decryption and data compression. Presentation layer is a best layer for cyptography.

Network Devices : Gateway Redirector operates on the presentation layer.

Layer 5 : Session Layer

Session layer establish and manages the session between the two users at different ends in a network. Session layer also manages who can transfer the data in a certain amount of time and for how long. The examples of session layers and the interactive logins and file transfer sessions. Session layer reconnect the session if it disconnects. It also reports and logs and upper layer errors.

Protocols : The protocols that work on the session layers are NetBIOS, Mail Slots, Names Pipes, RPC.

Network Devices : Gateway.

Layer 4 : Transport Layer

Transport layer manages end to end message delivery in a network and also provides the error checking and hence guarantees that no duplication or errors are occurring in the data transfers across the network. Transport layer also provides the acknowledgement of the successful data transmission and retransmits the data if no error free data was transferred.

It also provides and error handling and connectionless oriented data deliver in the network.

Protocols : These protocols work on the transport layer TCP, SPX, NETBIOS, ATP and NWLINK.

Network Devices : The Brouter, Gateway and Cable tester work on the transport layer.

Layer 3 : Network Layer

The network layer determines that how data transmits between the network devices. It also translates the logical address into the physical address *e.g.*, computer name into MAC address. It is also responsible for defining the route, managing the network problems and addressing. Router works on the network layer and if a sending device does not break the data into the similar packets as the receiving device then network layer split the data into the smaller units and at the receiving end the network layer reassembe the data.

Network layer routes the packets according to the unique network addresses. Router works as the post office and network layer stamps the letters (data) for the specific destinations.

Protocols : These protocols work on the network layer IP, ICMP, ARP, RIP, OSI, IPX and OSPF.

Network Devices : Network devices including Router, Brouter, Frame Relay device and ATM switch devices work on the network layer.

Layer 2 : Data Link Layer

Defines procedures for operating the communication links

Frames packets

Detects and corrects packets transmit errors

Protocols : Logical Link Control

- error correction and flow control
- manages link control and defines SAPs

802.1 OSI Model

802.2 Logical Link Control

Media Access Control

- communicates with the adapter card
- controls the type of media being used :

Layer 1 : Physical Layer

Physical layer defines and cables, network cards and physical aspects. It defines raw bit stream on the physical media. If also provides the interface between network and network communication devices. It is also responsible for how many volts for 0 and how many for 1. Physical layer also checks the number of bits transmitted per second and two ways or one way transmission. Physical layer also dealing with the optical, mechanical and electrical features.

Protocols : Protocols that work on the physical layer are ISDN, IEEE 802 and IEEE 802.2

Network Devices : Hubs, Repeaters, Oscilloscope and Amplifier work on the network devices.

15.2. INTRODUCTION TO ROUTING ALGORITHMS

Routing is the act of moving information across an internetwork from a source to a destination. Along the way, at least one intermediate node typically is encountered. Routing is often contrasted with bridging, which might seem to accomplish precisely the same thing to the casual observer. The primary difference between the two is that bridging occurs at Layer 2 (the link layer) of the OSI reference model, whereas routine occurs at Layer 3 (the network layer). This distinction provides routing and bridging with different information to use in the process of moving information from source to destination, so the two functions accomplish their tasks in different ways.

Routing is the decision making procedure by which one node selects one (or more) of its neighbours to forward a packet towards its ultimate destination.

- Routing-table computation
- Packet forwarding

Routing Table Computation : In computer networking a *routing table*, or **Routing Information Base (RIB)**, is an electronic table *(file)* or database type object that is stored in a router or a networked computer. The routing table stores the routes (and in some cases,

metrics associated with those routes) to particular network destinations. This information contains the topology of the network immediately around it. The construction of routing tables is the primary goal of routing protocols and static routes.

Routing tables and generally not used directly for packet forwarding in modern router architectures; instead, they are used to generate the information for a smaller forwarding table which contains only the routes which are chosen by the routing algorithm as preferred routes for packet forwarding, often in a compressed or pre-complicated format that is optimized for hardware storage and lookup.

The routing table consists of at least three information fields :

1. *the network id* ; i.e., the destination network id
2. *cost* : i.e., the cost or metric of the path through which the packet is to be sent
3. *next hop* ; The next hop, or gateway, is the address of the next station to which the packet is to be sent on the way to its final destination.

Depending on the application and implementation, it can also contain additional values that refine path selection :

1. *quality of service* associated with the router. For example, the U flag indicates that an IP routes is up.
2. *links to filtering criteria/access lists* associated with the route.
3. *inteface* : such as eth0 for the First Ethernet card, eth1 for the second Ethernet card etc.

Packet forwarding : Forwarding is the relaying of packets for one network segment to another by nodes in a computer network. The simplest forwarding model-unicasting - involves a packet being relayed from link to link along a chain leading from the packet's source to its destination. However, other forwarding strategies are commonly used. Broadcasting requires a packet to be duplicated and copies sent on multiple links with the goal of delivering a copy to every device on the network. In practice, broadcast packets are not forwarded everywhere on a network, but only to devices within a broadcast domain, making *broadcast* a relative term. Less common than broadcasting, but prehaps of greater utility and theoretical significance is multicasting, where a packet is selectively duplicated and copies delivered to each of a set of recipients.

Design Goals/Objectives for Good Routing Algorithm

Routing algorithms often have one or more of the following design goals :

1. Optimality
2. Simplicity and low overhead
3. Robustness and stability
4. Rapid convergence
5. Flexibility

1. Optimality : Optimality refers to the capability of the routing algorithm to select the t route, which depends on the metrics and metric weightings used to make the calculation. example, one routing algorithm may use a number of hops and delays, but it may weigh y more heavily in the calculation. Naturally, routing protocols must define their metric lation algorithms strictly.

2. Simplicity and Low overhead : Routing algorithms also are desig... as possible. In otherwords, the routing algorithm must offer its functio... with a minimum of software and utilization overhead. Efficiency is par... when the software implementing the routing algorithm must run on a comp... physical resources.

3. Robustness and Stability : Routing algorithms must be robust, which ... they should perform correctly in the face of unusual or unforeseen circumstances... hardware failures, high load conditions, and incorrect implementations. Because ... located at network junction points, they can cause considerable problems when the... The best routing algorithms are often those that have with stood the test of time and ... have proven stable under a variety of network conditions.

4. Convergence : In addition, routing algorithms must converge rapidly. Convergence is the process of agreement, by all routers, on optimal routers. When a network event causes routes to either go down or become available, routers distribute routing update messages that permeate networks, stimulating recalculation of optimal routes and eventually causing all routers to agree on these routes.

5. Flexibility : Routing algorithms should also be flexible, which means that they should quickly and accurately adapt to a variety of network circumstances. Assume, for example, that a network segment has gone down.

15.2.2. Routing Metrics

Routing algorithms have used many different metrics to determine the best route. Sophisticated routing algorithms can base route selection on multiple metrics, combining them in a single (hybrid) metric. All the following metrics have been used :

1. Path length
2. Reliability
3. Delay
4. Bandwidth
5. Load
6. Communication Cost

1. Path Length is the most common routing metric. Some routing protocols allow network administrators to assign arbitrary costs to each network link. In this case, path length is the sum of the costs associated with each link transversed.

2. Reliability in the context of routing algorithms, refers to the dependability (usually described in terms of the bit-error rate) of each network link. Some network links might go down more often than others.

3. Routing delay refers to the length of time required to move a packet from source to destination through the internet work. Delay depends on many factors, including the bandwith of intermediate network.

4. Bandwidth refers to the available traffic capacity of a link. All others things being equal, a 10-Mbps Ethernet link would be preferable to a 64-kbps leased line. Although bandwidth is a rating of the maximum attainable throughput on a link.

...ver to the degree to which a network resource, such as a router, is busy. Load can ...ed in a variety of ways, including CPU utilization and packet processed per second. ...communication cost is another important metric, especially because some companies ...ver care about performance as much as they care about operating expenditures. Although ...obby may be longer, they will send packets over their own lines rather than through the ...blic lines that cost money for usage time.

15.2.8. Destination Based Routing

- Optimal routing algorithm if the following is satisfied :
  - The cost of sending a packet P via a path is independent of the actual utilization of the path (load in involved).
  - The cost of concatenation of two paths equal the sum of the costs of the two paths :

    For all $i = 0, \ldots\ldots\ldots, k$

    $$C(\langle u_0, \ldots\ldots\ldots, u_k \rangle) = C(\langle u_0, \ldots\ldots\ldots, u_j \rangle) + C(\langle u_j + 1, \ldots\ldots\ldots, u_k \rangle)$$

  - The graph does not contain any cycle of negative cost.
- A path from u to v is optimal if there is no path from u to v with lower cost.

There are two algorithms based on destination based routing :

1. Shortest path routing
2. Destination Based routing

1. Shortest path routing : In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) such that the sum of the weights of its constituents edges is minimized. An example is finding the quickest way to get from one location to another on a road map; in this case, the vertices represent locations and the edges represent segments of road and are weighted by the time needed to travel that segment.

The problem is also sometimes called the **single-pair shortest path problem**, to distinguish it from the following generalizations :

- The single-source shortest path problem, in which we have to find shortest paths from a source vertex v to all other vertices in the graph.
- The single-destination shortest path problem, in which we have to find shortest paths from all vertices in the graph to a single destination vertex v. This can be reduced to the single-source shortest path problem by reversing the edges in the graph.
- The all-pairs shortest path problem, in which we have to find shortest paths between every pair of vertices v, v' in the graph.

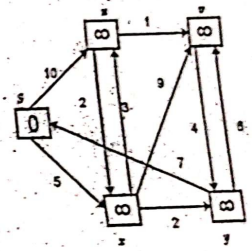
Dijkstra's algorithm is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge path costs, producing a shortest path tree Dijkstra's algorithm solves the single-source shortest-path problem when all edges have non-negative weights. It is a greedy algorithm and similar to Prim's algorithm. Algorithm starts at the source vertex, s, it grows a tree, T, that ultimately spans all vertices reachable from S. Vertices are added to T in order of distance *i.e.*, first S, then the vertex closest to S, then the next closest, and so on. Following implementation assumes that graph G is represented by adjacency list.

DIJKSTRA (G, w, s)

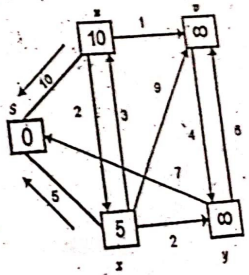
1. INITIALIZE SINGLE-SOURCE (G, s)
2. S ← { } // S will ultimately contains vertices of final shortest-path weights from s
3. Initialize priority queue Q *i.e.*, Q ← V[G]
4. while priority queue Q is not empty do
5. u ← EXTRACT_MIN(Q) //Pull out new vertex
6. S ← S ∪ { u }

   //Perform relaxation for each vertex v adjacent to u
7. for each vertex v is Ad[u] do
8. Relax (u, v, w)

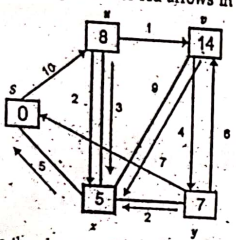
Step 1 : Given initial graph G = (V,E). All nodes have inifinite cost except the sourc node, s, which has 0 cost.



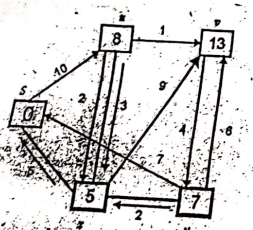
Step 2 : First we choose the node, which is close to the source node, s. We initialize d[to 0. Add it to S. Relax all nodes adjacent to source, s. Update predecessor (see red arrow i diagram below) for all nodes updated.



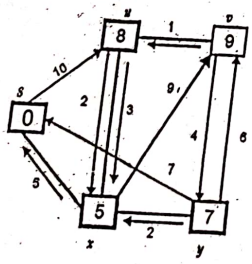
Step 3 : Choose the closest node, x. Relax all nodes adjacent to node x. Update predecessors for nodes u, v and y (again notice red arrows in diagram below).



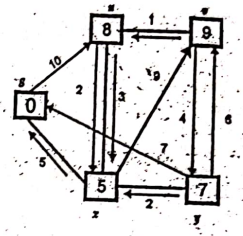
Step 4 : Now, node y is the closest node, so add it to S. Relax node v and adjust its predecessor.



Step 5 : Now we have node u that is closest. Choose this node and adjust its neighbour node v.



Step 6 : Finally, add node v. The predecessor list now defines the shortest path from each node to the source node, s.



Hence the shortest distance between node s and v is 8 and distance between node s and node y is 7.

15.3. ALL PAIR SHORTEST PATH PROBLEM (APP)

The *all-pairs shortest path* problem can be considered the mother of all routing problems. It aims to compute the shortest path from each vertex v to every other u. Using standard single-source algorithms, you can expect to get a naïve implementation of $O(n^3)$ if you use Dijkstra for example – *i.e.*, running a $O(n^2)$ process n times. Likewise, if you use the Bellman-Ford-Moore algorithm on a dense graph, it'll take about $O(n^4)$, but handle negative arc-lengths too.

Storing all the paths explicitly can be very memory expensive indeed, as you need one spanning tree for each vertex. This is often impractical in terms of memory consumption, so these are usually considered as all-pairs shortest distance problems, which aim to find just the distance from each to each node to another.

The all pair shortest pair problem is solved by Floyd Warshall algorithm.

15.3.1. Floyd Warshall Algorithm

- Let $d_{ij}^{(k)}$ be the weight of a shortest path from vertex i to vertex j for which all intermediate vertices are in the set {1, 2, ..., k}.
- When $k = 0$, a path from vertex i to vertex j with no intermediate vertex numbered higher than 0 has no intermediate vertices at all, hence $d_{ij}^{(0)} = w_{ij}$

$$d_{ij}^{(k)} = \begin{cases} \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k = 0 \\ w_{ij} & \text{if } k \geq 1. \end{cases} \quad \ldots(1)$$

Floyd-Warshall (*W*)

1. $n \leftarrow rows\ [W]$

$\leftarrow W$

$k \leftarrow 1$ to n

do for $i \leftarrow 1$ to n

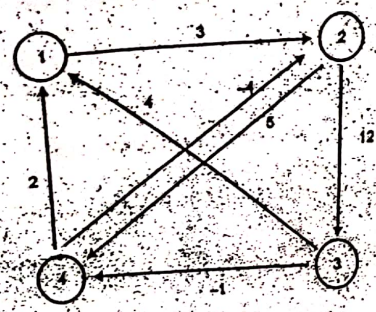
do for $j \leftarrow 1$ to n

do $d_{ij}^{(k)} \leftarrow \left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$

urn $D^{(n)}$;

ing time $O(V^3)$

ple : Consider a graph given below consisting of nodes 1, 2, 3 and 4. The respective e given in arcs. We will determine all the possible shortest path between different



$$D^0 = \begin{bmatrix} 0 & 3 & 0 & 0 \\ 0 & 0 & 12 & 5 \\ 4 & 0 & 0 & -1 \\ 2 & -4 & 0 & 0 \end{bmatrix} \qquad D^1 = \begin{bmatrix} 0 & 3 & 0 & 0 \\ 0 & 0 & 12 & 5 \\ 4 & 7 & 0 & -1 \\ 2 & -4 & 0 & 0 \end{bmatrix}$$

$$D^2 = \begin{bmatrix} 0 & 3 & 15 & 8 \\ 0 & 0 & 12 & 5 \\ 4 & 7 & 0 & -1 \\ 2 & -4 & 8 & 0 \end{bmatrix} \qquad D^3 = \begin{bmatrix} 0 & 3 & 15 & 8 \\ 16 & 0 & 12 & 5 \\ 4 & 7 & 0 & -1 \\ 2 & -4 & 8 & 0 \end{bmatrix}$$

$$D^4 = \begin{bmatrix} 0 & 3 & 15 & 8 \\ 7 & 0 & 12 & 5 \\ 1 & -5 & 0 & -1 \\ 2 & -4 & 8 & 0 \end{bmatrix}$$

D^0, D^1, D^2, D^3 and D^4 are all pair shortest path matrix for node 1 to node 4. The Floyd-Warshall algorithm compares all possible paths thorugh the graph between each pair of vertices. It is able to do this with only V^3 comparisions. This is remarkable considering that there may be up to V^2 edges in the graph, and every combination of edges is tested. It does so by incrementally improving an estimate on the shortest path between two vertices, until the estimate is known to be optimal.

15.4. DEADLOCK FREE PACKET SWITCHING

In packet switching networks the occurence of deadlock is very undesirable. Therefore one needs controllers (algorithms that control the flow of packets through the network), that prevent the occurrence of deadlock. We present a class of controllers that prevent deadlock, use only local information known to the processors and allow packets to have a variable size. Some controllers in this class are proven to be optimal with respect to other controllers using the same local information.

A (distributed) algorithm that permits some moves of the network and forbids others is called a controller. An important and very desirable property of controllers is that they prevent deadlock. A deadlock occurs if there is a situation when there are packets which will never arrive at their destination, no matter what sequence of moves is performed by the network.

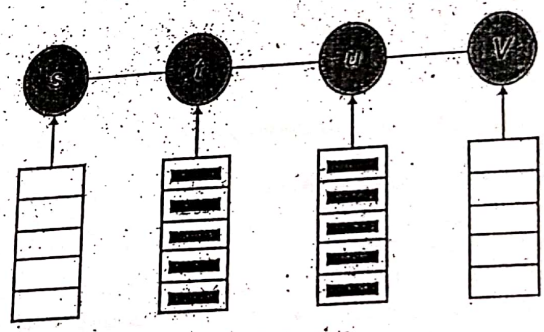


Fig. 15.2. Buffer-size 5

Node S sending 5 packets to V through t

Node V sending 5 packets to S through u .

Here each of the four has buffer B, each capable of containing exactly one packet node s has sent B packet with destination V to t and node v has sent B packet with destination s to u. All buffer in t and u are now occupied and consequently none of the packet stored in t and u can be forward toward its destination.

Packet Handling : The handling of packets by the node performed by 3 ways :

- Generation : A node u creates a new packet p and places it in an empty buffer in u. Node u is the source of P.

- *Forwarding* : A packet p is forwarded from a node u to an empty buffer in the next node w on its route.
- *Consumption* : A packet p occupying a buffer in its destination node is removed from the buffer.

The packet switching control has the following requirements :

1. The consumption of a packet (at its destination) is always allowed.
2. The generation of a paket in a node where all buffers are empty is always allowed.
3. The controller uses only local information, that is, whether a packet can be accepted in a node u depends only on information known to u or contained in the packet.

Solutions : There are two ways in which this problem is deal with :

- Structured solutions :
  - Buffer-graph based schemes
    - The destination schemes
    - The hops-so-far scheme
    - A cyclic orientation based scheme
- Unstructured solutions :
  - Forward count and backward count schemes
  - Forward state and backward state schemes

Buffer Graph

- *A buffer graph (for G, B)* is a directed graph BG on the buffers of the network, such that

1. BG is a cyclic (contains no directed cycle);
2. bc is an edge of BG if b and c are buffers in the same node, or buffers in two nodes connected by a channel in G; and
3. for each path $\pi \in P$ there exists a path in BG whose image is π.
   - P is the collection of all paths followed by the packets – this collection is determined by the routing algorithm.

The Destination Scheme

- Uses N buffers in each node u, with a buffer $b_u[v]$ for each possible destination V.
  - It is assumed that the routing algorithm forwards all packets with destination v via a directed tree T_v rooted towards v.

The buffer graph is defined by $BG = (B, E)$, where $b_u[v_1]\, b_w[v_2] \in E$ if $v_1 = v_2$ and uw is an edge of T_{v1}.

There exists a deadlock-free controller for arbitrary connected networks that uses N buffers in each node and allows packets to be routed via arbitrarily chosen sink trees.

The Hops-so-far Scheme

- Node u contains $k + 1$ buffers $b_u[0], \ldots\ldots, b_u[k]$.
- It is assumed that each packet contains a hop-count indicating how many hops the packet has made from its source.

The buffer graph is defined by $BG = (B, E)$, where $b_u[i]\, b_w[j] \in E$ if $i + 1 = j$ and uw is an edge of the network.

There exists a deadlock-free controller for arbitrary connected networks that uses $D + 1$ buffers in each node (where D is the diameter of the network), and requires packets to be sent via minimum-hop paths.

Acyclic Orientation Based Scheme

- An acyclic orientation of G is a directed acyclic graph obtained by directing all edges of G.
- A sequence $G_1, \ldots G_B$ of acyclic orientations of G is an acyclic orientation cover of size B for the collection P of paths if each path $\pi \in P$ can be written as a concatenation of B paths $\pi_1, \ldots\ldots \pi_B$, where π_i is a path of G_i.
  - A packet is always generated in node u in buffer $b_u[1]$
  - A packet in buffer $b_u[i]$ that must be forwarded to node w is placed in buffer $b_w[i]$ if the edge between u and w is directed towards w in G_i, and to $b_w[i + 1]$ if the edge is directed towards u in G_i.

If an acyclic orientation cover for P of size B exists, then there exists a deadlock-free controller using only B buffers in each node.

Forward and Backward-count Controllers

Forward-count Controllers :

- For a packet p, let s_p be the number of hops it still has to make to its destination ($0 \le s_p \le k$).
- For a node u, f_u denotes the number of free buffers in u ($0 \le f_u \le B$)

The controller accepts a packet p in nodes u if $s_p < f_u$.

If $B > k$ then the above controller is deadlock-free controller.

Backward-count Controllers :

- For a packet p, let t_p be the number of hops it has made for its source.

The controller accepts a packet p in node u if $t_p > k - f_u$.

15.5. ELECTION ALGORITHMS

An algorithm for choosing a unique process to play a particular role is called an *election algorithm*. For example, in a variant of our 'central-server' algorithm for mutual exclusion, the 'server' is chosen from among the processes p_i, $i = 1, 2 \ldots\ldots N$, that need to use the critical

...n electronic algorithm is needed to choose which of the processes will play the role ... It is essential that all the processes agree on the choice. Afterwards, if the process ...plays the role of server wishes to retire, then another election is required to choose a ...placement.

Each process $P_i (i = 1, 2 \ldots, N)$ has a variable $elected_i$ which will contain the identifier of the elected process. When the process first becomes a participant in an election it sets this variable to the special value '$\perp$' to denote that it is not yet defined.

Our requirements are that during any particular run of the algorithms :

E1 : (safety) A participant process p_i has $elected_i = \perp$ or $elected_i = P$, where P is chosen as the non-crashed process at the end of the run with the largest identifier.

E2 : (liveness) All process p_i participate and eventually set $elected_i \neq \perp$ – or crash.

15.5.1. A Ring-based Election Algorithm

We give the algorithm which is suitable for a collection of processes arranged in a logical ring. Each process p_i has a communication channel to the next process in the ring $p_{(i+1) \bmod N}$ and all messages are sent clockwise around the ring. We assume that no failures occur, and that the system is a synchronous. The goal of this algorithm is to elect a single process called the *coordinator*, which is the process with the largest identifier.

Initially, every process is marked as a *non-participant* is an election. Any process can begin an election. It proceeds by marking itself as a *participant*, placing its identifier in an *election* message and sending it to its clockwise neighbour.

When a process receives an *election* message, it compares the identifier in the message with its own. If the arrived identifier is the greater, then it forwards the message to its neighbour. If the arrived identifier is smaller and the receiver is not a *participant* then it substitutes its own indentifier in the message and forwards it; but it does not forward the message if it is already a *participant*. On forwarding an *election* message in any case, the process marks itself as a *participant*.

When a process p_i receives an *elected message*, it marks itself as a *non-particpant*, sets its variable $elected_i$ to the identifier in the message and, unless it is the new coordinator, forwards the message to its neighbour.

An example of a ring-based election in process is shown in Fig. 15.3. The *election* message currently contains 24, but process 28 will replace this with its identifier when the messages reaches it.

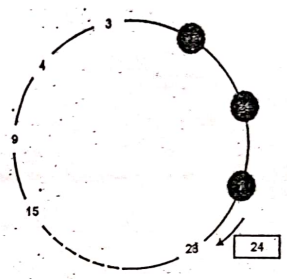


Fig. 15.3. A ring-based election in process

Note: The election was started by process 17. The highest process identifier encountered so far is 24. Participant processes are shown darkened.

While the ring-based algorithm is useful for understanding the properties of election algorithms in general, the fact that it tolerates no failures makes it of limited practical value. However, with a reliable failure detector it is in principle possible to reconstitute the ring when a process crashes.

15.5.2. Bully Algorithm

The bully algorithm allows processes to crash during an election, although it assumes that message delivery between processes is reliable. Unlike the ring-based algorithm, this algorithm assumes that the system is synchronous : it uses timeouts to detect a process failure. Another difference is that the ring-based algorithm assumed that processes have minimal *a priori* knowledge of one another : each knows only how to communicate with its neighbour, and none knows the identifiers of the other processes. The bully algorithm, on the other hand, assumes that each process knows which processes have higher identifiers, and that it can communicate with all such processes.

There are three types of message in this algorithm. An *election* message is sent to announce an election; an *answer* message is sent in response to an election message; and a *coordinator* message is sent to announce the identity of the elected process – the new 'coordinator'. A process beings an election when it notices, through timeouts, that the coordinator has failed. Several processes may discover this concurrently.

The process that knows it has the highest identifier can elect itself as the coordinator simply by sending a *coordinator* message to all processes with lower identifiers. On the other hand, a process with a lower identifier begins an election by sending an *election* message to those processes that have a higher identifier and awaits

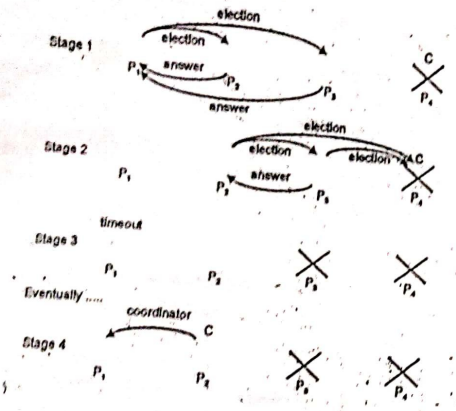


Fig. 15.4. The bully algorithm

The election of coordinator p_2 after the failure of p_4 and then p_3

The operation of the algorithm is shown in Fig. 15.4. There are four processes $p_1 - p_4$. Process p_1 detects the failure of the coordinator p_4, and announces an election (stage 1 in the figure). On receiving an *election* message from p_1, processes p_2 and p_3 send *answer* messages to p_1 and begin their own elections, p_3 sends an answer *message* to p_2 but p_3 receives no *answer* message from the failed process p_4 (stage 2). It therefore decides that it is the coordinator. But before it can send out the *coordinator* message, it too fails (stage 3). When p_1's timeout period T' *expires* (which we assume occurs before p_2's timeout expires), it deduces the absence of a *coordinator* message and begins another election. Eventually, p_2 is elected coordinator (stage 4).

If a process p_i receives a *coordinator* message, it sets its variable $elected_i$ to the identifier of the coordinator contained within it and treats that process as the coordinator.

If a process receives an *election* message, it sends back an *answer* message and begins another election – unless it has begun one already.

When a process is started to replace a crashed process, it begins an election. If it has the highest process identifier, then it will decide that it is the coordiantor and announce this to the other processes. Thus it will become the coordinator, even though the current coordinator is functioning. It is for this reason that the algorithm is called the 'Bully' Algorithm.

15.6. WAVE AND TRANSVERSAL ALGORITHM

A wave algorithm exchanges finite number of messages and then ... which depends causually on some event in each process.

- Used as a component in many applicaitons :
  - to synchronize nodes : synchronization algorithm
  - to disseminate information through a networks : broadcast
  - to compute a function that depends on information stored in all nodes of the network : converge cast - computation of a global function.

Wave algorithms are distributed algorithm that has 3 major requirements :

- Wave algorithms must satisfy the following 3 requirements :
  1. Termination : Each computation is finite :
     $\forall C : (|C| < \infty)$
  2. Decision : Each computation contians at least one decide event
     $\forall C : \exists e \in C$ e is a decide event
  3. Dependence : In each computation each decide event is causally preceded by an event in each process. (P is the set of computations)
     $\forall C : \exists e \in C : (e \text{ is the decide event} \Rightarrow \forall q \in P : \exists f \in C_q : f \leq e)$
- Wave algorithms can be characterised in several way :
  - Topology : An algorithm may be designed for a specific topology.
  - Centralisation : An algorithm is centralised if there is exactly one initiator.
  - Initial knowledge : Different algorithms requires different initial knowledge.
    - Process identity : Each process knows its unique name.
    - Neighbours identity : Each process knows the name of its neighbours.
    - Network characteristics : A process may know something about the network, *i.e.*, its diameter.
  - Complexity : Number of messages, size of each message.

15.6.1. Wave Algorithm

There are six general wave algorithms :

1. Ring Algorithm
2. Tree Algorithm
3. Echo Algorithm
4. Polling Algorithm
5. Phase Algorithm
6. Finn Algorithm

1. Ring algorithm :

- Processes are arranged in a unidirectional ring (each process has a sense of direction of knowledge of one dedicated neighbour.
- Initiator send mesage (tok) along the cycle.
- Each process passes it on.
- When it returns to initiator, initiator decides.

Theorem : ring algorithm is a wave algorithm.

For the initator

begin send (tok) to $Next_p$: receive (tok) : *decide* end

For the non-initators

begin receive (tok) ; send < tok > to $Next_p$ end

2. Tree Algorithm :

Operates on tree network (can work on spanning tree of arbitrary network)- no root, edges are undirected (bi-directional).

Tree Algorithm Properties :

- Termination
  - Each process sends at most 1 messsage.
  - N messages in computation C.
  - It takes finite time to reach a decision.
- Decision (at least one decision in a final configuration of C).
  - # of rec bits : $2(N-1)$
  - Processes that already sent a message : K
  - # of rec bits having false value : F
  - $F = (2N-2) - K$ (*)

The tree algorithm (for tree networks)

var $rec_p[q]$ for each $q \in Neigh_p$: boolean init false :

(* $rec_p[q]$ is true if p has received a message from q*)

begin while # [q : $rec_p[q]$ is false] > 1 do

begin receive <tok> from q : $rec_p[q]$: = *true* end

(*Now there is one q_0 for which $rec_p[q_0]$ is false*)

send <tok> to q_0 for which $rec_p[q_0]$ is false :

x : receive <tok> from q_0 : $rec_p[q_0]$: = true :

decide

(*Inform other processes of decision*)

for all $q \in Neigh_p$, $q \neq q_0$ do send <tok> to q

end

Echo Algorithm

* *Termination :*

- *Exactly one message on each incident edge*
- *For all statement : message are sent in parallel in 1 time unit*

- 2[E] message
- Correctness :
  - Decision means that a correct tree was built :
  - $pq \in E_x$; $father_p = q$
  - Prove that T does not contain a cycle!
  - Each process sends messages to all neighbours = >
    each process receives at least one message = >
    first (exactly one) message defines the father
- Dependance :
  - Front the algorithm and the connectivity of a network it is clear that messages from fathers and founds are delivered.
  - We need to prove that the backtracking works correctly (messages start to come from below)

The echo algorithm (networks of arbitrary topology)

var rec_p : integer init 0 : (*Counts number of received messages*)

$father_p$: P init def :

initator

being for all $q \in Neigh_p$ do send <tok> to q :

while $rec_p < \#\ Neigh_p$ do

begin receive <tok> ; rec_p : = rec_p 1 end

decide

end

non-initator

being receive <tok> from neighbour q : $father_p$: = q : rec_p : = $rec_p \neq 1$

For all $q \in Neigh_p$, $q \neq father_p$ do send <tok> to q :

while $rec_p < \#\ Neigh_p$ do

begin receive <tok> ; rec_p : = rec_p + 1 end :

send <tok> to $father_p$

end

Polling Algorithm : This algorithm is used in star network in which initiator is in centre. The initiator ask each neighbour to reply with message and decides after receipt of all the messages.

Var rec_p : integer int 0 : (*for initiator only*)

For the initator

begin for all $q \in Neigh_p$ do send <tok> to q :
while $rec_p < \#\ Neigh_p$ do
begin receive <tok> ; $rec_p := rec_p + 1$ end
decide
end

For non-initator :

begin receive <tok> from q : send <tok> to q end

Phase Algorithm : It is a decentralized algorithm for network of arbitrary topology. This can be used as wave algorithms for directed networks. The algorithm requires that the process known the diameter D of the network.

Applies to arbitrary (possibly directed) networks where the network diameter is known.

const D : integer = network diameter
var $Rec_p[q] : 0 \ldots D$ init 0, for each $q \in in_{p0}$ (* # msgs recvd from q*)
$sent_p : 0 \ldots D$ init 0, (* # msgs sent to each out-neighbour*)
begin if p is an initiator then
begin for all $r \in Out_p$ do send(tok) to r ;
$Sent_p := Sent_p + 1$;
end
while $min_q\ Rec_p[q] < D$ do
begin receive(tok) from q_1 ;
$Rec_p[q_1] := Rec_p[q_1] + 1$;
if $min_q\ Rec_p[q] \geq Sent_p$ and $Sent_p < D$ then
begin for all $r \in Out_p$ do send(tok) to r ;
$Sent_p := Sent_p + 1$;
end;
end;
decide
end

Finn's Algorithm :

- Applies to arbitrary (possibly directed) networks.

var Inc_p : set of processes init {p} ;
$NInc_p$: set of processes init {};

$rec_p[q]$: boolean for $q \in In_p$ init false;
begin if p is an initiator then
for all $r \in Out_p$ do send(Inc_p, $NInc_p$) to r;
while $Inc_p \neq NInc_p$ do
begin receive (I^*nc, $NInc$) for q;
$Inc_p := Inc_p \cup Inc$; $NInc_p := NInc_p \cup NInc_q$;
$rec_p[q] := true$;
if $\forall q_i \in In_p : rec_p[q_i]$ then $NInc_p := NInc_p \cup \{p\}$;
if Inc_p or $NInc_p$ has changed then
for all $r = Out_p$ do send (Inc_p, $NInc_p$) to r
end;
decide
end

15.6.2. Transversal Algorithm

- A traversal algorithm is an algorithm with the following three properties :
  - In each computation there is one initiator, which starts the algorithm by sending out exactly one message.
  - A process, upon receipt of a message, either sends out one message or decides.
  - The algorithm terminates in the initiator and when this happens, each process has sent a message at least once.

There are various traversal algorithm :

Senquential Polling : The pseudo code for this algorithm is given below :

var rec_p : integer init 0; // For initiator only
For the initiator
begin while $rec_p < \#\ Neigh_p$ do
begin send < tok > to $q_{rec_p + 1}$;
receive < tok >; $rec_p = rec_p + 1$
end;
decide
end
For non-initiators
begin receive < tok > from q; send < tok > to q; and

Classical depth first search .

- The classical depth-first search algorithm computes a depth-first search spanning tree using $2|E|$ message and $2|E|$ time units.

...ent three protocols are bidirectional protocols that belong to a class called sliding ... protocols. The three differ among themselves in terms of efficiency, complexity ...ffer requirements, as discussed later. In these, as in all sliding window protocols, ... outbound frame contains a sequence number, ranging from 0 upto maximum. The ...imum is usually $2^n - 1$ so the sequence number fits exactly in an n-bit field. the stop-and-wait sliding window protocols uses $n = 1$, restricting the sequence numbers to 0 and 1, but more sophisticated versions can be arbitrary n.

The essence of all sliding window protocols is that at any instant of time, the sender maintains a set of sequence number corresponding to frames it is permitted to send. These frames are said to fall within the sending window. Similarly, the receiver also maintains a receiving window corresponding to the set of frames it is permitted to accept. The sender's window and the receivers's window need not have the same lower and upper limits or even have the same size. In some protocols they are fixed in size, but in others they can grow or shrink over the course of time as frames are send and received.

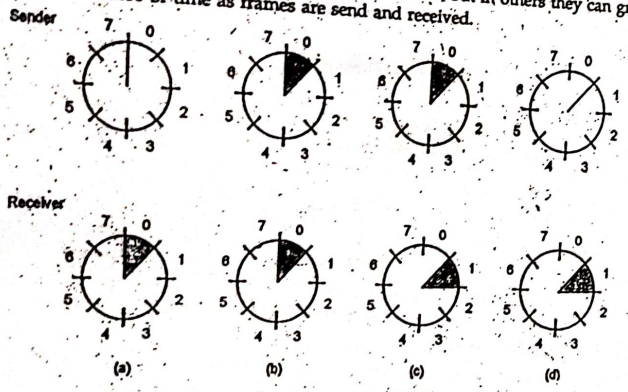


Fig. 15.5. A sliding window of size 1, with a 3-bit sequence number. (a) Initially, (b) After the first frame has been sent, (c) After the first frame has been received, (d) After the first acknowledgment has been received.

15.7.1. A One-Bit Sliding Window Protocol

Before tackling the general case, let us first examine a sliding window3 protocol with a maximum window size of 1. Such a protocol uses stop-and-wait since the sender transmits a frame and waits for its acknowledgment before sending the next one.

Figure 15.6, depicts such a protocol. Like the others, it start out by defining some variables *Next_frame_to_send* tells which frame the sender is trying to send. Similarly, *frame_expects* tells which frame the receiver is expecting. In both case, 0 and 1 are the only possibilities.

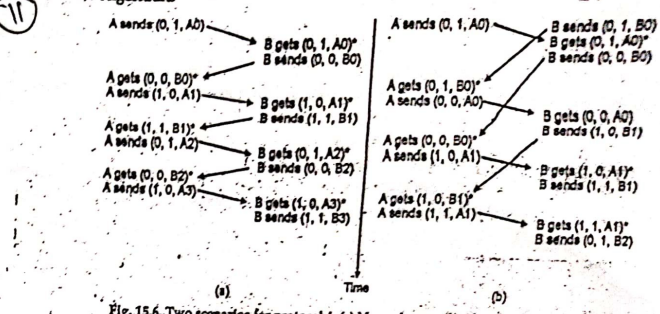


Fig. 15.6. Two scenarios for protocol 4. (a) Normal case, (b) Abnormal case. The notation is (seq, ack, packet number). An asterisk indicates where a network layers accepts a packet.

15.7.2. Protocol Using Go Back N

In Fig. 15.7 (a) we see go back n for the case in which the receiver's window is lar... Frame 0 and 1 are correctly received and acknowledged. Frame 2, however, is damaged lost. The sender, unaware of this problem, continues to send frame until the timer for fra... 2 expires. Then it backs up to frame 2 and starts all over with it, sending 2, 3, 4 etc. all ov... again.

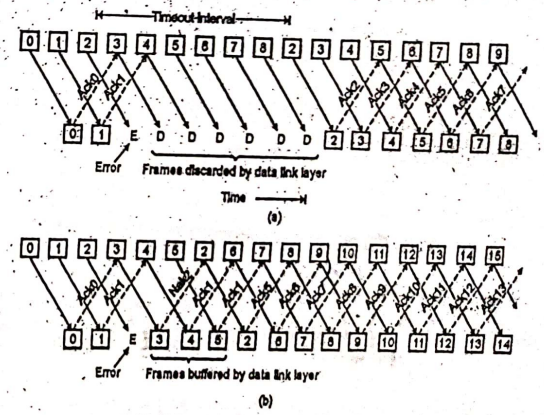


Fig. 15.7. Pipelining and error recovery. Effect of an error when (a) receiver's window size is 1 and (b) receiver's window size is large.

The other general strategy for handling errors when frames are pipelined is called selective repeat. When it is used, a bad frame that is received is discarded, but good frame received after it are buffered. When the sender times out, only the oldest unacknowledge frame is retransmitted. if that frame arrives correctly, the receiver can dellvered to the network layer, in sequence, all the frames it has buffered. Selective repeat is often combined with having the receiver send a negative acknowledgement (NAK) when it detects an error, for example, when it receivers a checksum error or a frame out of sequence. NAKs stimulated retransmlsion before the corresponding timer expires and thus improve performance.

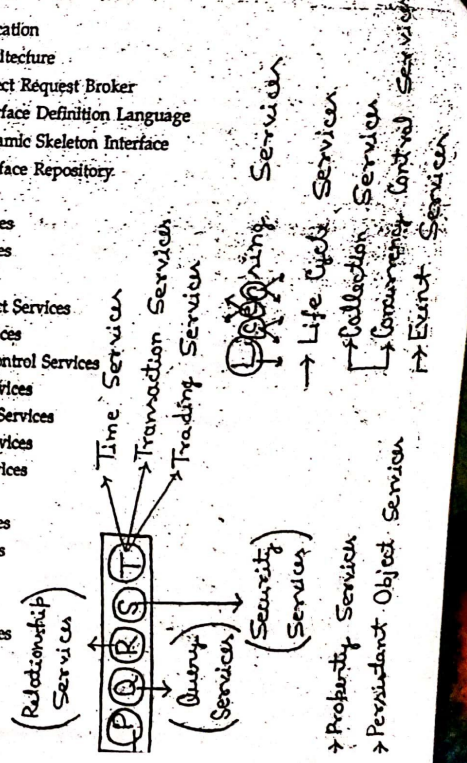
In Fig. 15.7 (*b*), frames 0 and 1 are again correctly received and acknowledged and frame 2 is lost. When frame 3 arrives at the receiver, the data link layer there notices that is has missed a frame, so it sends back a NAK for 2 but buffers 3. When frame 4 and 5 arrive, they, too, and buffered by the data link layer instead of being passed t the network layer. Eventually, the NAK 2 gets back to the sender, which immediately resends frame 2. When that arrives, the data link layer now has 2, 3, 4 and 5 and can pass all of them to the network layer.

REVIEW QUESTIONS

1. What is Routing? Explain destination based routing. *(UPTU : 2007, 2004, 2003)*
2. What are wave Algorithms? Discuss the usage and application of wave algorithms. What are the requirements of wave algorithm? *(UPTU : 2007, 05, 04)*
3. What is deadlock free packet switching? Explain approaches of DFPS.
4. Explain all pair shortest path problem with example.
5. What is election Algorithm? Explain the working Bully algorithm and how it is used to elect new coordinator. *(UPTU 2007)*
6. Explain functionalities of layers in OSI model.
7. Write short notes on
   (*i*) SMTP (*ii*) FTP
   (*iii*) HTTP (*iv*) DHCP.

□□□□

Chapter 16 CORBA: Case Study



start our study of distributed object-based systems by taking a look at the Common Request B roker Architecture, simply referred to as CORBA. As its name suggests, CORBA is not so much a distributed system but rather the specification of one. These specifications have been drawn up by the Object Management Group (OMG), a nonprofit organization with over 800 members, primarily from industry. An important goal of the OMG with respect to CORBA was to define a distributed system that could overcome many of the interoperability problems with integrating networked applications. The first CORBA specifications became available in the beginning of the 1990s. At present, implementations of CORBA version 2.4 are widely deployed, whereas the first CORBA version 3 systems are becoming available.

Like many other systems that are the result of the work of committees, CORBA has features and facilities in abundance. The core specifications consist of well over 700 pages, and another 1,200 are used to specify the various services that are built on top of that core. And naturally, each CORBA implementation has its own extensions because there is always something that each vendor feels cannot be missed but was not included in the specifications. CORBA illustrates again that making a distributed system that is simple may be a somewhat overwhelmingly difficult exercise.

16.1 OVERVIEW OF CORBA

The architecture of CORBA adheres to a reference model of the OMG that was [illegible]. This reference model, shown in Fig. 16.1, consists of four groups of elements organized around what is called the Object Request Broker (ORB). The ORB is the core of every CORBA distributed system. It is responsible for enabling communication between objects and their clients while hiding issues related to distributed processing. Roughly speaking, the ORB is implemented as libraries that are linked with a client and server application, and that offers basic communication services. We return to the ORB below when discussing CORBA's object model.

Besides objects that are built as part of specific applications, the reference model also distinguishes what are known as CORBA facilities. Facilities are constructed as compositions

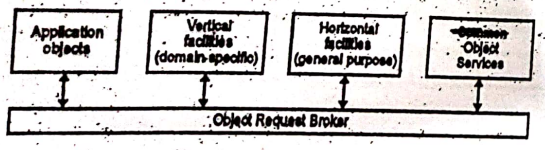


Fig. 16.1. The global architecture of CORBA

of CORBA services (which we discuss below), and are split into two different groups. Horizontal facilities consist of general-purpose high-level services that are independent of application domains.

16.2. OBJECT MODEL

CORBA uses the remote-object model that was described in Chapter 2. In this model the implementation of an object resides in the address space of a server. It is interesting t note that the CORBA specifications never explicitly state that objects should be implemente only as remote objects. However, virtually all CORBA systems support only this model. I addition, the specifications often suggest that distributed objects in CORBA are actuall remote objects. Later, when discussing the Globe object model, we show how a completel different model of an object could, in principle, be equally well supported by CORBA.

Objects and services are specified in the CORBA Interface Definition Language (IDL CORBA IDL is similar to object interface definition languages in that it provides a precis syntax for expressing methods and their parameters. It is not possible to describe semantics i CORBA IDL. An interface is a collection of methods, and objects specify which interfaces the implement.

Interface specifications can be given only by means of IDL. As we shall see later, in system such as Distributed COM and Globe, interfaces are specified at a lower level in the form c tables. These so-called binary interfaces are by their nature independent of any programmin language. In CORBA, however, it is necessary to provide exact rules concerning the mappin of IDL specifications to existing programming languages. At present, such rules have bee given for a number of languages, including C, C++, Java, Smalltalk, Ada and COBOL.

Given that CORBA is organized as a collection of clients and object servers, the gener organization of a CORBA system is shown in Fig. 16.2.

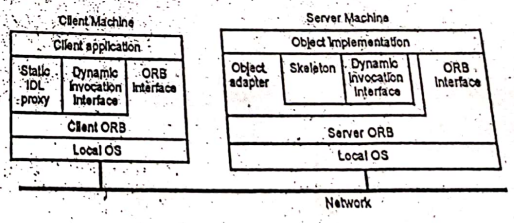


Fig. 16.2. The general organization of a CORBA system.

Underlying any process in CORBA, be it a client or server, is the ORB. The ORB ca best be seen as the runtime system that is responsible for handling the basic communicatio between a client and an object. This basic communication consists of ensuring that a invocation is sent to the object's server and that the reply is passed back to the client.

From the perspective of a process, the ORB offers only a few services itself. One c these services is manipulating object references. Such references are generally dependent o

a particular ORB. An ORB will therefore offer operations to marshal and unmarshal object references so that they can be exchanged between processes, as well as operations for comparing references. Object references are discussed in detail below.

Other operations offered by an ORB deal with initially finding the service that are available to a process. In general, it provides a means to obtain an inital reference to an object implementing a specific CORBA service. For example, in order to make use of a naming service, it is necessary that a process knows how that to refer to that service. These initialization aspects apply equally well to other services.

Interface and Implementation Repository : To allow the dynamic construction of invocation requests, it is important that a process can find out during runtime what an interface looks like. CORBA offers an interface repository, which stores all interface definitions. In many systems, the interface repository is implemented by means of a separate process offering a standard interface to store and retrieve interface definitions. An interface repository can also be viewed as that part of CORBA that assists in runtime type checking facilities.

Besides an interface repository, a CORBA system generally offers also an implementation repository. Conceptually, an implementation repository contains all that is needed to implement and activate objects. Because such functionality is intimately related to the ORB itself and the underlying operating system, it is difficult to provide a standard implementation.

Whenever an interface definition is compiled, the IDL compiler assigns a repository identifier to that interface. This repository ID is the basic means to retrieve an interface definition from the repository. The identifier is by default derived from the name of the interface and its methods, implying that no guarantees are given with respect to its uniqueness. If uniqueness is required, the default can be overridden.

Introduction

The Common Object Request Broker Architecture (CORBA) from the Object Management Group (OMG) provides a platform-independent, language-independent architecture for writing distributed, object-oriented applications. CORBA objects can reside in the same process, on the same machine, down the hall, or across the planet. The Java language is an excellent language for writing CORBA programs. Some of the features that account for this popularity include the clear mapping from OMG IDL to the Java programming language, and the Java runtime environment's built-in garbage collection.

16.3. OMG CORBA SPECIFICATION

What is CORBA ? : CORBA, or Common Object Request Broker Architecture, is a standard architecture for distributed object systems. It allows à distributed, heterogeneous collection of objects to interoperate.

The OMG : The Object Management Group (OMG) is responsible for defining CORBA. The OMG comprises over 700 companies and organizations, including almost all the major vendors and developers of distributed object technology, including platform, database and application vendors as well as software tool and corporate developers.

16.3.1. CORBA Architecture

CORBA defines an architecture for distributed objects. The basic CORBA paradigm is that of a request for services of a distributed object. Everything else defined by CORBA is in terms of this basic paradigm.

The services that an object provides are given by its *interface*. Interfaces are described in OMG's Interface Definition Language (IDL). Distributed objects are identified by object references, which are typed by IDL interfaces.

The OMG's Object Management Architecture (OMA) tries to define the various high-level facilities that are necessary for distributed object-oriented computing. The core of the OMA is the Object Request Broker (ORB), a mechanism that provides object location transparency, communication, and activation. Based on the OMA, the CORBA specification which provides a description of the interfaces and facilities that must be provided by compliant ORBs was released.

CORBA is composed of five major components : ORB, IDL, dynamic invocation interface (DII), interface repositories (IR), and object adapters (OA). These are discussed in the following sections.

16.3.1.1. The Object Request Broker

The CORBA specification must have software to implement it. The software that implements the CORBA specification is called the ORB. The ORB, which is the heart of CORBA, is responsible for all the mechanisms required to perform these tasks :

- Find the object implementation for the request.

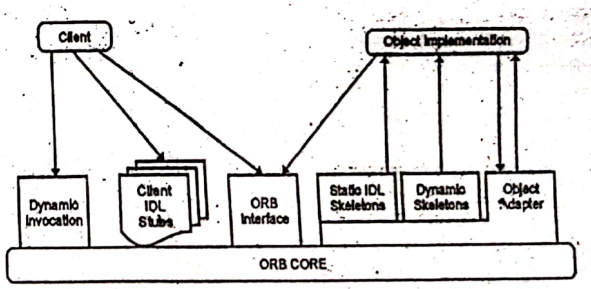


Fig. 16.3. The structure of the CORBA 2.0 ORB

Different vendors and different ORBs : Since there is more than one CORBA implementation, and these implementations are from different vendors, a good question at this point would be whether objects implemented in different ORBs from different vendors would be able to communicate with each other.

...rface Definition Language

...with RMI, CORBA objects are to be specified with interfaces, which are the contract ...the client and server. In CORBA's case, however, interfaces are specified in the special ...tion language IDL.

The IDL defines the types of objects by defining their interfaces. An interface consists of a set of named operations and the parameters to those operations. Note that IDL is used to describe interfaces only, not implementations. Despite the face that IDL syntax is similar to C++ and Java, IDL is not a programming language.

Dynamic invocation interface : Invoking operations can be done through either static or dynamic interfaces. Static invocation interfaces are determined at compile time, and they are presented to the client using stubs. The DII, on the other hand, allows client applications to use server objects without knowing the type of those objects at compile time. It allows a client to obtain an instance of a CORBA object and make invocations on that object by dynamically constructing request.

16.3.1.3. Dynamic Skeleton Interface

Analogous to the DIII is the server-side dynamic skeleton interface (DSI), which allows servers to be written without having skeletons, or compile-time knowledge, for the objects being implemented.

16.3.1.4. Interface Repository

The IR provides another way to specify the interfaces to objects. Interfaces can be added to the interface respository service. Using the IR, a client should be able to locate an object that is unknown as compile time, find information about its interface, then build a request to be forwarded through the ORB.

Object adapters : An object adapter is the primary way that an object implementation accesses services provided by the ORB. Such services include object reference generation and interpretation, method invocation, security of interactions and object and implementation activation and deactivation.

16.4. CORBA SERVICES

The various services provided by CORBA are listed below :

1. Naming Service
2. Security Service
3. Event Service
4. Persistent Object Service
5. Life Cycle Service
6. Concurrency Control Service
7. Notification Service
8. Externalization Service
9. Relationship Service
10. Transaction Service
11. Query Service
12. Licensing Service
13. Property Service
14. Time Service
15. Trading Service
16. Collection Service

CORBA Services
The Object Management Architecture

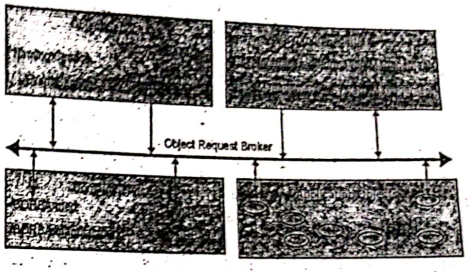


Fig. 16.4. CORBA Services

16.4.1. Naming Service

- Most Commonly used Service in CORBA
  - Provides the principle way for clients to find objects on the network
  - Remote object references can be bound to names
  - Clients may access object references by providing the name
- A Naming Context represents a directory or subdirectory of named objects
  - This interface can be used to bind, lookup, or unbind objects and subcontexts within the naming directory.
  - Names of objects within Naming Context are composed of Name Component arrays
  - Browsing can be done with a Binding Iterator

16.4.2. Security Service

- Provides the tools to secure distributed applications
  - Authentication, access control for users
  - Secure communication channels
- High-level security framework
  - Implementations are free to use any cryptographic framework
  - Layers security measures on top of ORB object-to-object model
- Interfaces :
  - Principle Authenticator, Credentials object assigned to each user
  - Current object identifies security measures for current execution context
  - Extensions to the org.omg.CORBA.Object interface

16.4.3. Event Service

- Flexible framework for asynchronous object interactions
  - DII provides basic form of asynchrony
  - Event Service targeted to *notification*, rather than *interaction*
- Event Service allows to set up event channels
- CORBA objects may act as
  - Event consumers
  - Event suppliers
- Push or pull event communication
- Interfaces :
  - Push Consumer, Pull Consumer, Push Supplier, Pull Supplier, Event Channel

16.4.4. Persistent Object Services

- Common framework to interact with persistency eng. :
  - Relational databases, object databases, etc.
  - Middle ware between CORBA objects and database protocols (ODMG)
- Persistency ....
  - Managed at object level or data member level
  - Objects typically control their own persistent state.
- Interfaces :
  - PO (Persistent object), PID (persistent object identifier), POM (persistent object manager)
- Persistent object service depends on :
  - Externalization Service, Life Cycle Service

16.4.5. Life Cycle Service

- Standard protocols for distributed objects :
  - Creation, copying, movement, deletion of remote objects.
- Service defined around the concept of object factories
- Interfaces
  - Life Cycle Object
  - Factory Finder interface for locating object factories
- Life Cycle Service references Naming Service
  - When dealing with connected graphs of objects, the Relationship Service structures are referenced.

16.4.6. Concurrency Service

- Framework for managing concurrent object access
  - Analogous to multithreading support in C++/Java
  - Facilities for interfacing with transaction service

- Concurrency Service assumes usage of locks
  - Read, write-locks, multi-possession, two-phase locks
  - Conflicts with existing locks are resolved by a first-come first-served
- Interfaces to represent resources;
  - LockSet, Transactional LockSet
  - LockSet Factory Interface for object creation

16.4.7. Collection Service

- Grouping of objects
  - Sets, queues, sequences
  - Iterators for these collections
  - Factory classes for object creation
- Collection Interface
  - Set, Heap, Stack, Queue, SortedSet and subclasses
- Iterator Interface
  - Equality Iterator, Sequential Iterator are subclasses
- Operations Interface
  - Base class for operations on objects

16.4.8. Externalization Service

- Means for Object conversion
  - Allow export over general media (network streams, disk storage, etc.)
  - Constitute data back into object references (Potentially in a different ORB/process)
- Pluggable data formats for externalized objects
  - Standard serialized format for objects is provided
  - Streaming model for externalizing and internalizing objects
- Interfaces :
  - Stream, StreamFactor, File StreamFactory, etc.;
  - Objects must extend streamable interface
- Externalization Service uses Life Cycle Service

16.4.9. Licensing Service

- Controlled access to objects and services under a licensing model
  - Conceptually an extension of security service
- Operation
  - Client requests licensed service, proofs ownership of license

...ce provider checks with license manager
...ce provider may request notification on license expiration of many poll for ...anged in license state.
...terfaces
– License Server Manager, Producer Specific License Service
• License Service depends on Security Service

16.4.10. Notification Service

• Extends asynchronous message exchange of Event Service
– Allows multiple event suppliers to send events to multiple event consumers
– Supports pull and push models
– Allows event channels to be federated
– Allows clients to attach filters to each proxy in an event channel
• QoS properties :
– Per-channel, per-proxy, per-event
• CORBA Notification Service uses :
– Structured Event, Event Channel, Event Type classes
– Structured Push Consumer, Structured Push Supplier classes

16.4.11. Property Service

• Defines name/value pairs that can be assigned to objects
– Without being explicitly defined by their IDL interfaces
– Can represent any application-specific attributes
• Property Service does not specify how properties are associated with objects
– Implementation detail
– Properties are represented as string name and Any value
• Interfaces
– Property Set, Property Set Def (inquire metadata about properties-read/write; read-only, etc.)
– Properties Iterator, Property Set Factory

16.4.12. Query Service

• General query mechanism for distributed objects
– Collections of objects can be searched to generate sub collections
– Subsets of objects within a collection can be deleted of updated
• Query Service's facilities can be mapped to persistent storage facilities
– Relational databases, object databases
• Interfaces :
– Collection objects (with an Iterator), Collection Factory
– Query Manager, Query Evaluator
– Result of a query typically is a Collection object

16.4.13. Relationship Service

• Allows for explicit specification of realtionships among objects
– Defined in terms of type, roles within the relationship and the cardinality of each role
– Objects fulfill a role when they participate in a relationship
– Agent/proxy relationship : one agent, multiple proxies
• Interfaces :
– Relationship, Role
– Relationship Factory, Role Factory, Relationship Iterator
– Cos Graphs, Cos Containment, Cos Reference

16.4.14. Time Service

• Ability to enquire accurate time value + estimated error
– Uses Universal Coordinated time representation
– Time intervals of 100 nano seconds since Oct 15, 1582
– Times are relative to Greenwich Time Zone
• Time-based events, linear positioning of events
– Implementation of time service is responsible for communication with accurate time source (Cesium clock, radio time broadcast, etc.)
• Interfaces :
– Time Service object, UTO (Universal time objects)
– Time EventT, Timer Event Service, Timer Event Handler
• Timer event portion depends on Event service

16.4.15. Trading Object Service

• Market trading context
– Objects describes services offered to the system
– Clients issue description of desired service
– Trading service performs matching
• Interfaces :
– Lookup interface to advertise needs of importers
– Register interface to advertise properties of a service
– Offer Iterator to iterate through multiple offers (hits)
– Admin interface to query for all outstanding offers and queries and to control matching process.

16.4.16. Transaction Service

• Defines interfaces to allow distributed objects to create and engage in transactional interactions

- ACID properties
  - Atomic : any and all actions carried out as part of a transaction are committed of undone/cancelled
  - Consistent : actions within a transaction produce results that are consistent
  - Isolated : transactions do not see each other's effects until they are committed. If they are rolled back, their effects are not seen by other contexts.
  - Durable : if a transaction completes successfully, its effects are made persistent.
- Transaction can involve a series of remote method calls
  - Whole transaction is rolled back when a significant error is encountered
  - Transaction contexts are propagated along the way
- Service provides framework for notification and management of transaction boundaries.
  - Little help with implementation of rollback operations.
- Interfaces :
  - Current interface : start and end of transactions
  - Control interface : manipulation of ongoing transactions
  - Terminator, coordinator, Resource objects
- Depends on Concurrency and Persistent Object Services

REVIEW QUESTIONS

1. What is CORBA ? Explain various CORBA Services. *(UPTU – 2008, 2007, 2005, 2004)*
2. Explain why the interfaces to remote objects in general and CORBA objects in particular do not provide constructions. Explain how CORBA objects can be created in absence of constructions. *(UPTU - 2008)*
3. Explain OMG architecture of CORBA.
4. Explain the diagram, the architecture of CORBA. Mention the use of vendors and ORB's in architecture.
5. Write short notes on
   (*i*) Naming Services
   (*ii*) Event Services
   (*iii*) Transaction Services.

□□□□

PREVIOUS YEARS QUESTION P

B.Tech

Semester VIII Examination, 2003-04

Distributed System

Time : 3 Hours *Total Ma*

Note : Attempt All questions.

1. Attempt any *Four* parts :

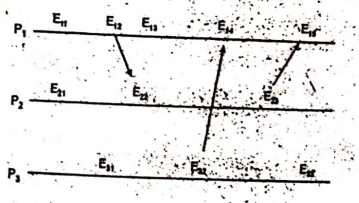
(*a*) What are Distributed systems ? What are significant advantages and limitati of Distributed system ? Explain with the example, what could be impact of absen of global clock and shared memory.

(*b*) What are the fundamental issues in Resource Management in Distributed system Explain the difference between Data Migration, Computation Migration and Distributed Scheduling.

(*c*) What are Lamport's Logical Clocks ? List the important conditions to be satisfied by Lamport Logical Clocks. If A and B represent two distinct events in a process and if $A \rightarrow B$ then $C(A) < C(B)$ but vice versa is not true. Explain the reasons.

(*d*) What do you mean by Casual Ordering of Message ? If process P sends two messages M_1 and M_2 to another process Q what problem may arise if the two messages are not received by recipient Q in the order they were sent by process P ? Develop an algorithm which guarantees the casual ordering of message in Distributed system.

(*e*) What are Vector Clocks ? What are the advantages of Vector over Lamport clock? For the space time diagram shown below, obtain the vector time stamp of various events.



(*f*) What do you mean by Distributed Mutual Exclusion ? A simple solution to this problem may be obtained by having a designated site called control site which receives all requests and grants permission to execute critical section. Discuss the limitation of this approach. Discuss any protocol, which uses the time, stamp to order critical section request and resolve conflict in simultaneous request for critical section execution.