B. TECH. TCS-801/TIT-801

EIGHTH SEMESTER EXAMINATION, 2008-2009

DISTRIBUTED SYSTEM

Time : 3 *Hours* *Total Marks :* 100

Note : 1. *Attempt all the questions.*
2. *All questions carry equal marks.*

1. Answer any four parts of the following : (5 × 4 = 20)

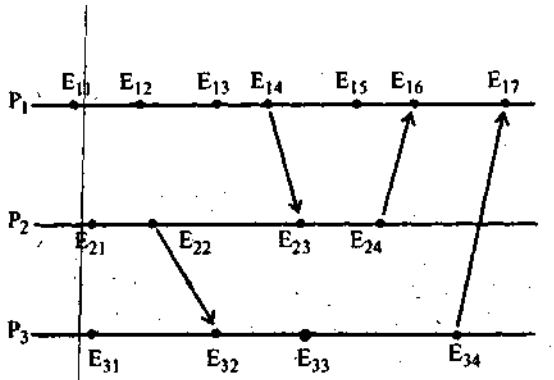
(a) What are distributed systems? What are significant advantages and limitations of distributed systems? Explain with the example, what could be the impact of absence of global clock and shared memory?

Ans. Please see Q. No. 1(a) of 2003-04.

(b) Why is scalability an important feature in the design of distributed system? Discuss some of the guiding principles for designing a scalable distributed system.

Ans. Scalability is a desirable property of a system, a network or a process, which indicates its ability to either handle growing amounts of work in a graceful manner or to be readily enlarged. For example, it can refer to the capability of a system to increase total throughput under an increased load when resources (typically hardware) are added. An analogous meaning is implied when the word is used in a commercial context, where scalability of a company implies that the underlying business model offers the potential for economic growth within the company. The design of applications, as well as the design of next generation middleware systems must follow a set of guiding principles in order to ensure long-term "survivability" without costly re-engineering. From our practical experience, the key determinants to success in this endeavour are adherence to the following principles: (1) Design for change; (2) Provide for storage subsystem I/O coordination; (3) Employ workload partitioning and load balancing techniques; (4) Employ caching; (5) Schedule the work load; and (6) Understand the work load. In order to support these principles, a large collection extensive experimental results are collected comparing three middleware systems targeted at data and compute-intensive applications implemented by some research groups during the course of the last decade, on a single data and compute-intensive application. The main contribution of this work is the analysis of a level playing field, where we discuss and quantify how adherence to these guiding principles impacts overall system throughput and response time.

(c) What are Vector Clocks? Explain with the help of implementation rule of Vector Clocks, how they are implemented. What are the advantages of Vector Clock over Lampart Clock? For the space time diagram shown below, obtain the vector time stamp of various events.



Ans. Vector clock : Let n be the number of processes in a distributed system. Each process p_i is equipped with clock c_i which is an integer vector of length n.

The clock c_i may be a function that assign a vector c_i (a) to any event a. c_i (a) is referred to as the timestamp of event c_i at p_i . c_i $[i]$, the i^{th} entry of c_i correspond to p_i, own logical time $c_i[j]$ $j \neq i$ is p_i's best guess of the logical time at p_j.

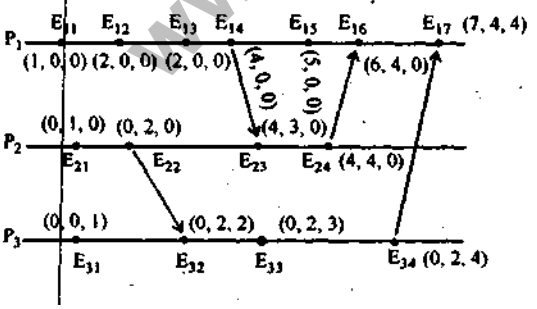
The implementation rules for the vector clock are

IR 1 : Clock c_i is incremented between any two successive event in process p_i.

$$c_i[i] := c_i[i] + d \quad (d > 0)$$

IR 2 : If event a is sending message m by process p_i then message m is assigned a vector timestamp $t_m = c_i$ (a); on receiving the same message m by process the same message m by process p_j, c_j is updated as follows

$$\forall k,\ c_j[k] := \max (c_j[k], t_m[k])$$



(d) What do you mean by Global state of the distributed system? What are the differences between consistent Global state, Transitless Global State and Strongly Consistent Global State?

Ans. The global state of a distributed computation is the set of local states of all individual processes involved in the computation plus the state of the communication channels by which we can synchronized our system. Global states can be used for distributed/parallel application monitoring and control. Strongly Consistent Global States (SCGS) are especially well suited for on-line controlling. Existing SCGS detecting algorithms work with process local states. They must wait for state terminations before complete states can be taken into account. Because of that the global states seen by a monitor always belong to the past. We present an algorithm, which works with unterminated local states. This approach lets the monitor detect SCGS sooner: currently lasting global states can be perceived promptly after they started. Application control based on SCGS detection should react quicker to arising situations when using the new algorithm. The quick reactions contribute to a better parallel/ distributed application performance. Simulation tests confirm these suppositions. Our solution utilizes bounded maximal message transfer time. It is compared with another method, which employs frequent confirmation messages. While both methods can lead to similar application performance, our approach induces a few times lower both monitor load and network traffic.

Consistent Global State : A consistent global state (CGS) contains pairwise concurrent local states

- Consistent Global states :

  – a global state GS = {$LS1$, $LS2$, ... LSn} is consistent iff

$\forall i, \forall j : 1 \leq i, j \leq n ::$ inconsistent (LS_i, LS_j) = ϕ

- Transitless global states :

  – a global state GS = {LS1, LS2, ... LSn} in transitless iff

  $\forall i, \forall j : 1 \leq i, j <= n ::$ transit (LSi, LSj,) = ϕ

  Strongly consistent global states :

  – It is consistent and transitless

(e) What do you mean by casual ordering of message? If process P sends two messages M1 and M2 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.

Ans. Please see Q. No 1(d) of year 2003-04

(f) What do you mean by problem of Mutual Exclusion in distributed system? What are the requirements of a good mutual exclusion algorithm? Explain the performance metrics to judge the performance of distributed mutual exclusion algorithm.

Ans. A distributed algorithm is presented that realizes mutual exclusion among N nodes in a computer network. The algorithm requires at most N message exchanges for one mutual exclusion invocation. Accordingly, the delay to invoke mutual exclusion is smaller than in an algorithm of Ricart and Agrawala, which requires 2*(N – 1) message exchanges per invocation. A drawback of the algorithm is that the sequence numbers contained in the messages are unbounded. It is shown that this problem can be overcome by slightly increasing the number of message exchanges. Categories and Subject Descriptors: C.2A [Computer Systems Organization): Computer-Communication Networks-distributed systems; D.4.1 [Operating Systems]: Process Management mutual exclusion.

Requirements of Mutual Exclusion Algrithms : The primary objective of a mutual exclusion algorithm is to maintain mutual exclusion; that is to guarantee the only one request accesses the CS at a time. In addition the following characteristics are considered important in a mutual exclusion algorithm:

Freedom from Deadlocks. Two or more sites should not endlessly wait indefinitely to execute CS while other sites are repeatedly executing CS. That is, every requesting site should get an opportunity to execute CS in a finite time.

Fairness. Fairness dictates that requests must be executed in the order they are made (or the order in which they arrive in system). Since a physical global clock does not exist, time is determined by logical clocks. Note that fairness implies freedom from starvation, but not vice-versa.

Fault Tolerance. A mutual exclusion algorithm is fault-tolerant if in the wake of a failure, it can recorganize itself so that it continues to function without any prolonged disruptions.

How to Measure the Perforemance : The performance of mutual exclusion algorithms is generally measured by the following four matrics: First, the number of messages necessary per CS invocation. Second, the synchronization delay, which is the time required after a site leaves the CS and before the next site enters the CS. Note that normally one or more sequential message exchanges are required after a site exits the CS and before the next site enters the CS. Third the response time, which is the time interval a request waits for its CS exection to be over after its request messages have been sent out. Thus, response time does not include the time a request waits at a site before its request

messagges have been sent out. Fourth, the system throughput, which is the rate at which the system executes requests for the CS. If *sd* is the synchronization delay and *E* is the average critical section execution time, then the throughput is given by the following equation:

$$\text{system throughput} = 1/(sd + E)$$

2. Attempt any two of the following : (10 × 2 = 20)

(a) Explain how the two-phase commit protocol for nested transactions ensures that if the top-level transaction commits, all the right descendants are committed or aborted.

Ans. The two-phase commit protocol (2PC), sometimes referred as TPC is a distributed algorithm that lets all nodes in a distributed system agree to commit a transaction. The protocol results in either all nodes committing the transaction or aborting, even in the case of network failures or node failures. However, the protocol will not handle more than one random site failure at a time. The two phases of the algorithm are the commit-request phase, in which the coordinator attempts to prepare all the cohorts, and the commit phase, in which the coordinator completes the transactions.

(b) What are agreement protocols? What are Byzantine agreement problem, the consensus problem and Interactive Consistency Problem?

Ans. Agreement Protocols : The agreement protocols allow non faulty processors to reach a comman agreement in the distributor system wheather there are other processes faulty or not. The common agrement among the processor is taken through the agreement protocol. The processor setting the common agreement require the exchange of message among various sites of distributed system many times. These sites communicate with each other about a common agreement through the agreement protocol.

The Byzantine Agreement Problem : In the Byzantine agreement problem, an arbitrarily chosen processor, called the source processor, broadcasts its initial value to all other processors. A solution to the Byzantine agreement problem should meet the following two objectives :

Agreement. All non-faulty processors agree on the same value.

Validity. If the source processor is non-faulty, then the common agreed upon value by all non-faulty processors should be the initial value of the source.

Table : The three agreement problems

Problem →	**Byzantine Agreement**	**Consensus**	**Interactive Consistency**
Who initiates the value	One processor	All processors	All processors
Final agreement	Single value	Single value	A vector of values

Two points should be notes :

1. If the source processor is faulty, then all non-faulty processors can agree on any common value.
2. If is irrelevant what value faulty processors agree on or whether they agree on a value at all.

The Consensus Problem : In the consensus problem, every processor broadcasts its initial value to all other processors. Initial values of the processors may be different. A protocol for reaching consensus should meet the following conditions :

Agreement All non-faulty processors agree on the same single value.

Validity If the initial value of every non-faulty processor is v.

If the initial values of non-faulty processors are different, then all non-faulty processors agree on.

The Interactive Consistency Problem : In the interactive consistency problem, every processor broadcasts its initial value to all other processors. The initial values of the processors may be different. A protocol for the interactive consistency problem should meet the following conditions :

Agreement : All non-faulty processors agree on the same vector, $(v_1, v_2, ..., v_n)$.

Validity : If the *i*th processor is non-faulty and its initial value is v_i, then the ith value to be agreed on by all non-faulty processors must be v_i.

If the jth processor is faulty, then all non-faulty processors can agree on any common value for v_j. It is irrelevant what value faulty processors agree on.

(c) What is the problem of distributed deadlock detection ? What are the differences in Centralized, Distributed and Hierarchical control organizations for distributed deadlock detection? What are advantages of distributed control organization over centralized control organization for distributed deadlock detection?

Ans. Please see Q. No. 2(a) of year 2003-04.

3. Attempt any two of the following : **(10 × 2 = 20)**

(a) What do you mean by distributed objects? Explain the concept of remote method invocation with a suitable example. How are the parameters and results passed to a remote procedure? Explain with a suitable example.

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

The term may also generally refer to one of the extensions of the basic object concept used in the context of distributed computing, such as replicated objects or live distributed objects.

Replicated objects are groups of software components (replicas) that run a distributed multi-party protocol to achieve a high degree of consistency between their internal states, and that respond to requests in a coordinated manner. Referring to the group of replicas jointly as an object reflects the fact that interacting with any of them exposes the same externally visible state and behaviour.

Live distributed objects (or simply live objects) generalize the replicated object concept to groups of replicas that might internally use any distributed protocol, perhaps resulting in only a weak consistency between their local states. Live distributed objects can also be defined as running instances of distributed multi-party protocols, viewed from the object-oriented perspective as entities that have distinct identity, and that can encapsulate distributed state and behaviour RMI allows a Java program to invoke a method that is being executed on a remote machine.

```
import java.rmi. *;
    public interface ReceiveMessageInterface extends Remote
                                {
     void receiveMessage(String x) throws RemoteException;
                                }
                        import java.rmi.*;
                    import java.rmi.registry.*;
                     import java.rmi.server.*;
                        import java.net.*;
public class RmiServer extends java.rmi.server.UnicastRemoteObject
                implements ReceiveMessageInterface
                                {
                            int thisPort;
                          String this Address;
Registry registry; // rmi registry for lookup the remote objects.
  // This method is called from the remote client by the RMI.
 // This is the implementation of the "ReceiveMessageInterface".
   public void receiveMessage(String x) throws RemoteException
                                {
                        System.out.println(x);
                                }
            public RmiServer() throws RemoteException
                                {
                                try{
                  // get the address of this host.
    this Address = (InetAddress.getLocal Host()). toString();
                                }
                          catch(Exception e){
      throw new RemoteException("can't get inet address.");
                                }
          thisPort = 3232; // this port(registry's port)
           System.out.println("this address="+this Address
                        +",port="+thisPort);
                                try{
       // create the registry and bind the name and object.
         registry = LocateRegistry.createRegistry( thisPort);
                  registry.rebind("rmiServer", this);
                                }
                      catch(RemoteException e){
                              throw e;
```

```
}
}
static public void main(String args[])
{
try{
RmiServera=new RmiServer();
}
catch(Exception e){
e.printStackTrace();
System.exit(1);
}
}
}
```

RmiClient.java

```
import java.rmi.*;
import java.rmi.registry.*;
import java.net.*;
public class RmiClient
{
static public void main(String args[])
{
ReceiveMessageInterface rmiServer;
Registry registry;
String serverAddress=args[0];
String serverPort=args[1];
String text=args[2];
System.out.println("sending"+text+"to
"+serverAddress+":"+serverPort);
try{
//get the "registry"
registry=LocateRegistry.getRegistry(
serverAddress,
(new Integer(serverPort)).intValue()
);
// look up the remote object
rmiServer=
(Receive Message Interface) registry.lookup ("rmiServer"));
// call the remote method
rmiServer.receiveMessage(text);
}
```

```
        catch (RemoteException e){
            e.printStackTrace();
        }
        catch (NotBoundException e){
            e.printStackTrace();
        }
    }
}
```

(b) 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?

Ans. The Andrew File System (AFS) is a distributed networked file system which uses a set of trusted server to present a homogeneous, location-transparent file name space to all the client workstations. It was developed by Carnegie Mellon University as part of the Andrew Project. It is named after Andrew Carnegie and Andrew Mellon. Its primary use is in distributed computing.

AFS has several benefits over traditional networked file systems, particularly in the areas of security and scalability. It is not uncommon for enterprise AFS cells to exceed twenty five thousand clients. AFS uses Kerberos for authentication, and implements access control lists on directories for users and groups. Each client caches files on the local file system for increased speed on subsequent requests for the same file. This also allows limited file system access in the event of a server crash or a network outage.

Read and write operations on an open file are directed only to the locally cached copy. When a modified file is closed, the changed portions are copied back to the file server. Cache consistency is maintained by a mechanism called callback. When a file is cached the server makes a note of this and promises to inform the client if the file is updated by some one else. Callbacks are discarded and must be re-established after any client, server, or network failure, including a time-out. Re-establishing a callback involves a status check and does not require re-reading the file itself.

A consequence of the file locking strategy is that AFS does not support large shared databases or record updating within files shared between client systems. This was a deliberate design decision based on the perceived needs of the university computing environment. It leads, for example, to the use of a single file per message in the original email system for the Andrew Project, the Andrew Message System, rather that a single file per mailbox.

A significant feature of AFS is the volume, a tree of files, sub-directories and AFS mountpoints (links to other AFS volumes). Volumes are created by administrators and linked at a specific named path in an AFS cell. Once created, users of the filesystem may create directories and files as usual without concern for the physical location of the volume. A volume may have a quota assigned to it in order to limit the amount of space consumed. As needed, AFS administrators can move that volume to another server and disk location without the need to notify users; indeed the operation can occur while files in that volume are being used.

AFS volumes can be replicated to read-only cloned copies. When accessing files in a read-only volume, a client system will retrieve data from a particular read-only copy. If at some point that copy becomes unavailable, clients will look for any of the remaining copies. Again, users of that data are unaware of the location of the read-only copy; administrators can create and relocate such copies as needed. The AFS command suite guarantees that all read-only volumes contain exact copies of the original read-write volume at the time the read-only copy was created.

The file name space on an Andrew workstation is partitioned into a shared and local name space. The shared name space (usually mounted as /afs on the Unix file-system) is identical on all workstations. The local name space is unique to each workstation. It only contains temporary files needed for workstation initialization and symbolic links to files in the shared name space.

The Andrew File System heavily "influenced Version 4 of Sun Microsystems' popular Network File System (NFS). Additionally, a variant of AFS, the Distributed File System (DFS) was adopted by the Open Software Foundation in 1989 as part of their Distributed computing environment.

Network File System (NFS) is a network file system protocol originally developed by Sun Microsystems in 1984 allowing a user on a client computer to access files over a network in a manner similar to how local storage is accessed. NFS, like many other protocols, builds on the Open Network Computing Remote Procedure Call (ONC RPC) system. The Network File System is an open standard defined in RFCs, allowing anyone to implement the protocol.

(c) **What is a digital signature? What are its uses in the security of a distributed system? Give a method to create a digital signature. Describe how digital signature can be used for ensuring message integrity in a distributed system.**

Ans. A digital signature or digital signature scheme is a type of asymmetric cryptography. For messages sent through an insecure channel, a properly implemented digital signature gives the receiver reason to believe the message was sent by the claimed sender. Digital signatures are equivalent to traditional handwritten signatures in many respects; properly implemented digital signatures are more difficult to forge than the handwritten type. Digital signature schemes in the sense used here are cryptographically based, and must be implemented properly to be effective. Digital signatures can also provide non-repudiation, meaning that the signer cannot successfully claim they did not sign a message, while also claiming their private key remains secret; further, some non-repudiation schemes offer a time stamp for the digital signature, so that even if the private key is exposed, the signature is valid nonetheless. Digitally signed messages may be anything representable as a bitstring: examples include electronic mail, contracts, or a message sent via some other cryptographic protocol.

Digital signatures are often used to implement electronic signatures, a broader term that refers to any electronic data that carries the intent of a signature but not all electronic signatures use digital signatures. In some countries, including the United States, and in the European Union, electronic signatures have legal significance. However, laws concerning electronic signatures do not always make clear whether they are digital cryptographic signatures in the sense used here, leaving the legal definition, and so their importance, somewhat confused.

A digital signature scheme typically consists of three algorithms :

- A *key generation* algorithm that selects a private key uniformly at random from a set of possible private keys. The algorithm outputs the private key and a corresponding public key.

- A *signing* algorithm which, given a message and a private key, produces a signature.
- A *signature verifying* algorithm which given a message, public key and a signature, either accepts or rejects.

Two main properties are required. First, a signature generated from a fixed message and fixed private key should verify on that message and the corresponding public key. Secondly, it should be computationally infeasible to generate a valid signature for a party who does not possess the private key.

Creation of digital signatures : First, you need a digital ID card called a digital certificate. Digital certificates, obtained by authorized third parties such as Verisign and Thawte, are essential for creating digital signatures.

Signing button in the MS Outlook : The following is a list of certifying authorities from which you can get a digital ID and read how-to for your software:

Thawte Certification : Thawte Certification offers free personal certificates for signing and encrypting email, these certificates are recognized and trusted by the majority of email clients that are in use on the Internet at present. Thawte is a global CA that has already certified 30% of the world's Internet e-commerce servers.

Verisign is a leading provider of digital authentication products and services. Through a special offer from Verisign, email users can obtain a free trial digital ID that you can use to positively identify yourself to, or receive encrypted messages from, business associates, friends, and online services when you use secure email.

GlobalSign is a Certification Authority (CA) that, with its private key issues, signs and manages digital certificates. The policy and the procedures GlobalSign uses for this are incorporated in the Certification Practices of GlobalSign (CPS).

BT offers secure server certificates for companies with websites and intranets, and personal digital certificates for the MS Outlook Express and Outlook. These certificates are issued under the VeriSign Global Trust Network, allowing global interoperability across intranet, extranet and Internet applications.

Message integrity : In many scenarios, the sender and receiver of a message may have a need for confidence that the message has not been altered during transmission. Although encryption hides the contents of a message, it may be possible to change an encrypted message without understanding it. (Some encryption algorithms, known as nonmalleable ones, prevent this, but others do not.) However, if a message is digitally signed, any change in the message after signature will invalidate the signature. Furthermore, there is no efficient way to modify a message and its signature to produce a new message with a valid signature, because this is still considered to be computationally infeasible by most cryptographic hash functions.

4. Attempt any two of the following : **(10 × 2 = 20)**

(a) Explain why serial equivalence requires that once a transaction has released a lock on an object, it is not allowed to obtain any more locks.

A server manages the objects a_1, a_2 ..... a_n. The server provides two operations for its clients:

read (i) returns the value of a_i

write (i, value) assigns value to a_i

The transaction T and U are defined as follows :

T : x = read (i); write (j, 44);

U : Write (i, 55); write (j, 66);

Describe an interleaving of the transaction

T and U in which locks are released early with the effect that the interleaving is not serially equivalent.

Ans.

(b) What are locks? What are essential differences in the lock-based protocols and Time stamp-based protocols?

Ans. Please see Q. No 4(b) of year 2004-05

(c) What are commit protocols? Explain how two-phase commit protocols respond to failure of participating site and failure of co-ordinator.

Ans. Please see Q. No 4(c) of year 2003-04

5. Answer any two of the following :

(a) What are the differences in centralized and distributed algorithms? How is the performance of a distributed algorithm evaluated? Explain the term, message complexity in reference to distributed algorithm.

Ans. Centralized ADS solutions employ a device in a data center near a server or server cluster. The device intercepts traffic passing to and from the server(s), and directs and/or modifies the intercepted traffic. Modifications to intercepted server traffic must be understood on the user's end, so the data center device must communicate with client software that makes sense of the modifications. The user's browser serves as the most ubiquitous standard client; therefore, at present Centralized ADS solutions are typically deployed to deliver to Web-based applications.

Some Centralized ADS vendors provide proprietary software clients that can further accelerate Web applications as well as optimize non-Web applications. These clients provide a cost-effective alternative to Distributed ADS solutions in home offices or "micro branches".

Distributed ADS solutions rely on a device in the data center and companion devices in remote offices. These devices are placed near WAN ingress/egress points where they can see, prioritize, and modify traffic. Because Distributed ADS solutions require access to the remote office, they are limited to private or virtual private networks. In the case of telecommuting or mobile workers, Distributed ADS vendors sometimes supply the "remote device" as software installed on the user's PC.

A critical difference between these two approaches is where and how they can be applied. Another important aspect of the two approaches is that the centralized approach is inherently open and interoperable, while distributed solutions are closed and vendor specific. You can buy Centralized ADS solutions from vendors A and B as long as they operate in front of different servers. The users will continue to use the same browser to access all "enhanced" applications.

However, if you buy a Distributed ADS solution from vendors D and E, they will not interoperate. Thus to experience the full benefit of solutions D and E, you must install both in all locations, which can be costly. Furthermore, some features of D may adversely affect the work of E. Operating two different distributed solutions is tricky, and at best they work as "ships in the night" ignoring each other.

The bottom line is that you can be a multi-vendor Centralized ADS shop, but you will typically be forced to adopt a single-vendor Distributed ADS solution.

Performance of a distributed algorithm is evaluated by various parameters :

1. **Incremental growth :** Computing power can be added in small increments.
2. **Reliability :** If one machine crashes, the system as a whole can still survive.
3. **Speed :** A distributed system may have more total computing power than a mainframe.
4. **Open system :** This is the most important point and the most characteristic point of a distributed system.

Since it is an open system it is always ready to communicate with other systems. An open system

that scales has an advantage over a perfectly closed and self-contained system.

Message complexity refers to the total number of messages exchanged by the algorithm.

(b) Explain why the interfaces to remote objects in general and CORBA objects in particular do not provide constructors. Explain how CORBA objects can be created in the absence of constructors.

Ans. CORBA (Common Object Request Broker Architecture) 2.0 1994 Dec.

Middleware by OMG (Object Management Group)

Microsoft: DCOM (Distributed Component Object Model)

IDL (Interface Definition Language)

ORB (Object Request Broker)

Object Management Architecture : (OMA)

1. **Object Request Broker :** define CORBA object bus
2. **CORBA Services :** define the system-level object frameworks that extend the bus
3. **CORBA Facilities :** define horizontal and vertical application frameworks that are used directly by business objects
4. **Application Objects :** the business objects and application

ORB functions :

1. static and dynamic method invocations
2. high-level language bindings
3. self-describing system
4. local/remote transparency
5. build-in security and transactions
6. polymorphic messaging
7. coexistence with existing systems

ORB client side :

1. the client IDL stubs
2. the Dynamic Invocation Interface (DII)
3. the interface repository APIs
4. the ORB interface

ORB server side :

1. server IDL stubs: skeleton
2. the Dynamic Skeleton Interface(DSI)
3. the object adapter
4. the implementation repository
5. the ORB interface

Note : global identifier: repository ID

IIOP : Internet Inter-ORB Protocol

ESIOP : Environment-specific Inter-ORB Protocol

GIOP : General Inter-ORB Protocol

IOR : Interoperable Object References

CORBA Services :

1. Life Cycle Service
2. Persistence Service
3. Naming Service
4. Event Service
5. Concurrent Control Service
6. Transaction Service
7. Relationship Service
8. Externalization Service
9. Query Service
10. Licensing Service
11. Properties Service
12. Time Service
13. Security Service
14. Trader Service
15. Collection Service
16. Start Up

Note : OpenDoc: DDCF (Distributed Document Component Facility)

Note : Business Objects consists of :

Business objects; Business process objects; Presentation objects

- **3-tier Client/Server, Object-Style**

  Tier 1 : View objects

  Tier 2 : Server objects

  Tier 3 : Legacy applications

- **The evolution of Web Technology**

  1994 : Hypertext Web: graphic Web, Browsers and Hyperlinks

  1995-1996 : Interactive Web: Forms, tables, CGI (Common Gateway Interface)

  Secured Transactions: SSL, S-HTTP, Firewalls, E-cash

  1997-1998 : Object Web:

  Java Web : Applets, Mobile components

  Distributed Objects : Orblets, compound documents, ActiveXs, CORBA" Cyberdog, Java Beans, Java Jars

- **Object Web Model Based on Java clients and CORBA ORBs**

  Tier 1 : Web client (HTML & Forms) → HTTP → Tier 2: Web Server → CORBA IIOP → (HTML documents) (CGI Apps)

  → Internet (TCP/IP) → (CORBA, ORB, Business object)

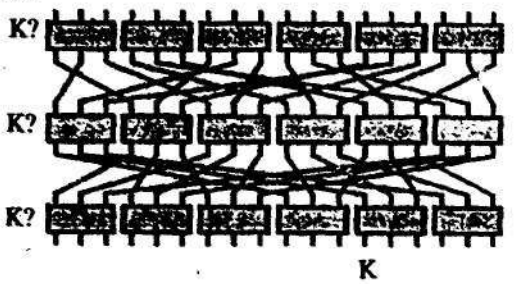
  → Tier 3: DBMS, Lotus Notes, TP monitors

- **Client/Server interactions on the Object Web**
  1. Web browser downloads HTML page
  2. Web browser retrieves Java applet from HTTP server
  3. Web browser loads applet
  4. Applet invokes CORBA server objects
- **CORBA brings to JAVA**
1. CORBA avoids the CGI bottleneck
2. CORBA provides a scalable server-to-server infrastructure
3. CORBA extends JAVA with a distributed object infrastructure
- **JAVA brings to CORBA**
  1. JAVA allows CORBA to move intelligent behaviour around
  2. JAVA complements the CORBA life cycle and Externalization Services
  3. JAVA is making CORBA ubiquitous on the web
  4. JAVA simplifies code distribution in large CORBA system
  5. JAVA complements CORBA's agenting infrastructure
  6. JAVA complements CORBA's component service
  7. JAVA is a great language for writing CORBA objects
- **CORBA static method invocations :** From IDL to interface Stubs
  1. Define your object classes using IDL
  2. Run the IDL file through a language precompiler
  3. Add the implementation code to the skeleton
  4. Compile the code
  5. Bind the class definitions to the Interface Repository
  6. Register the run-time objects with the Implementation Repository
  7. Instantiate the object on the server

(c) Write short notes on :

(i) Destination based routing.

Ans.



A packet enters the second switching module in the top row. The module looks up the destination address, "K", in its routing table and determines that the packet should exit out of left-most port. The packet travels to the third module in the second row. This module looks up "K" in its routin table and determines that the packet should exit out the middle port. The packet then travels to the fifth module in the

last row. The module looks up "K" in its routing table and determines that the packet should exit out of the left port.

The disadvantages of destination-based routing is that requires a table lookup at each switching module and that global information is required to compile the table.

The advantages of destination-based routing is that it can handle almost any topology and that it can also handle fault induced change in the topology.

(ii) Deadlock Free Packet Switching

Ans. Model :

- The network is a graph G = (V, E)
- Each node has B buffers

Moves :

- **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.

Requirements : The packet switching controller has the following requirements :

1. The consumption of a packet (at its destination) is always allowed.
2. The generation of a packet 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 depends only on information known to or contained in the packet.

Solutions :

- Structured solutions
  - Buffer Buffer-graph based schemes
- The destination scheme
- The hops hops-so so-far scheme
- Acyclic 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 B is a directed graph) BG on the buffers of the network, such that
  - BG is acyclic (contains no directed cycle);
  - *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
  - for each path $\pi \forall\_$ *P* there exists a path in BGBG whose image whose is π.
  - P is the collection of all paths followed by the packets this collection is determined by the routing algorithm.

Forward and Backward Backward-count Controllers : Forward Forward-count Controller Controller

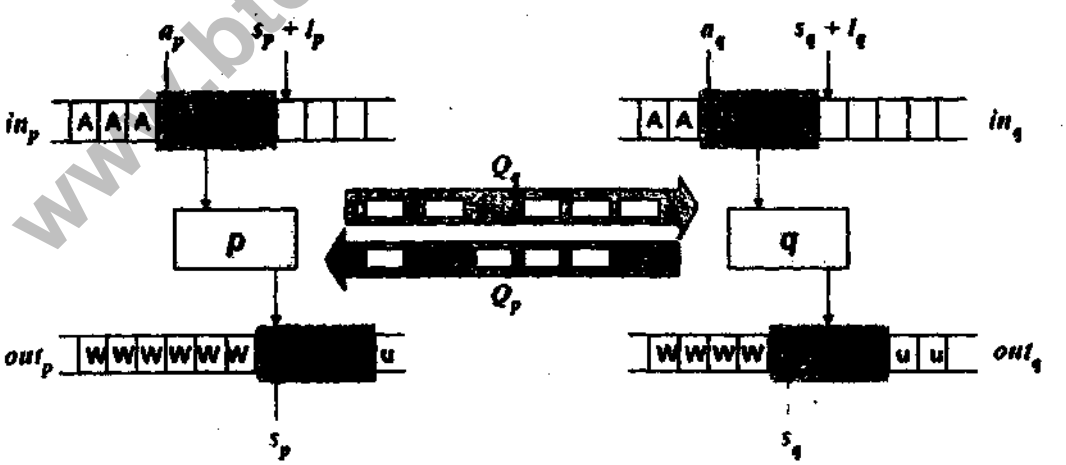
- For a packet p, let sp be the number of hops it still has to make to its destination ($0 << sp << kk$)
- For a node u, fu denotes the number of free buffers in u ($0 << fu << B$))
  The controller accepts a packet p in node u iff $sp < f\ fu$.
  If $B > k$ then the above controller is a deadlock deadlock-free controller
  Backward Backward-count Controller Controller:
- For a packet p, let tp be the number of hops it has made from its source.
  The controller accepts a packet p in node u iff $tp > k - fu$.

(iii) Balanced sliding window protocol.

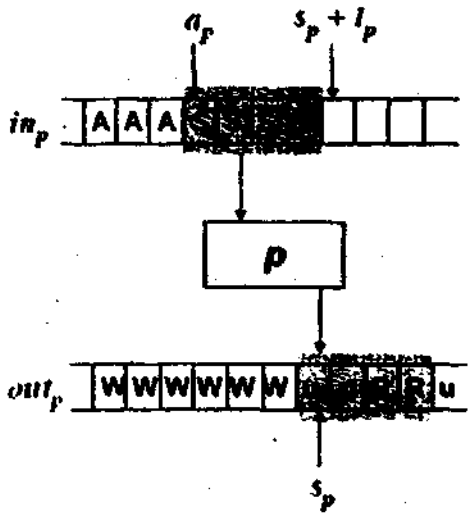
Ans. Two processes, p and q, each sending an infinite array of words to the other.
For Process p :
in inp : : An infinite array of words to be sent to process q
out outp : : An infinite array of words being received from process q
Initially for all i, out outp [i] = udef
sp : : The lowest numbered word that p still expects to receive from q
At any time, p has already written out outp [0] [through out outp [i]
Required Properties safe delivery : In every reachable configuration of the protocol
out [0... 1] = in [0... 1] and outp[sp – inq[sp-
out [0... 1] = in [0... 1] outq[sq - inp[sq- Eventual delivery
For every integer $k >> 00$ a configuration with, $sspp >> kk$ and $ssqq >> kk$ is eventually reached
delivery: – delivery: –
The protocol :

- The packet, < pack, w, i > ; transmits the word w = in inp[[i] to q.
- The processes use constants lp and lq as follows:
  - Process p can send the word w = *in inp* [[i] (as the packet, < pack, w, i >) only after storing all the words out outp [0]
    [through out outp [[i -lp], that is, , $i < s\ sp + lp$.
  - When p receives < pack, w, i >, retransmission of words, from in inp[0] [through in inp [[i – lq] is no longer necessary.

The Sliding Windows



The protocol is



S_p : $\{a_p \le i < S_p + l_p\}$
begin send < pack, $in_p[i]$, i > to q end

R_p : $\{< pack, w, i > \in Q_p\}$
begin receive < pack, w, i > ;
if out_p $[i]$ = *udef* then
begin $out_p[i] = w$;
a_p = max $\{a_p, i - l_q + 1\}$
sp = min $\{j \mid out_p [j]$ = undef$\}$
end
// else ignore – retransmission
end

L_p : $\{< pack, w, i > \in Q_p\}$
begin $Q_p = Q_p \setminus \{< pack, w, i >\}$ end

The protocol invariant is :

P= $\forall i < s_p : out_p[i] \neq$ udef

$\wedge \forall i < s_q$: out_q $[i] \neq$ udef
$\wedge <$ pack, w, $i > \in Q_p$
$\Rightarrow$ $w = in_q[i] \wedge (i < S_p + l_p)$
$\wedge <$ pack, w, $i > \in Q_q \Rightarrow w = in_p[i] \wedge (i < S_p + l_p)$
$\wedge$ $out_p[i] \neq$ udef $\Rightarrow$ $out_p[ii] = in_q$ $[i] \wedge (a_p > i - l_q)$
$\wedge$ $out_q[i] \neq$ udef $\Rightarrow$ $out_p[i] = in_p$ $[i] \wedge (a_q > i - l_p)$
$\wedge a_p < s_q$
$\wedge a_q < s_p$

Result is Safety : The protocol satisfies the requirement of safe delivery

Liveness :

- *P implies sp – lq << aapp << ssqq << aaqq + + llpp << sspp + + llpp*
- P implies that the sending of < pack, in inp $[s_q]$, s_q > by p or the sending of < pack, in $inq[s_p]$, s_p > by > q is applicable.
  Hence no deadlock is possible
- The protocol satisfies the requirement of eventual delivery