B.TECH.

EIGHT SEMESTER EXAMINATION, 2007-2008

DISTRIBUTED SYSTEMS

Time – 3 hours | Total Marks – 100

Note: (1) Attempt all questions.

(2) All questions carry equal marks.

Q. 1. Attempt any four parts of the following:-

Q. (a). What are distributed system? Explain its challenges in brief.

Ans. Distributed System: It deals with hardware and software systems containing more than one processing element or storage element, concurrent processes, or multiple programs, running under a loosely or tightly controlled regime.

In distributed computin a program is split up into parts that run simultaneously on multiple computers communicating over a network. Distributed computing is a form of parallel computing, but parallel computing is most commonly used to describe program parts running simultaneously on multiple processors in the same computer.

Challenges summarized as follows:

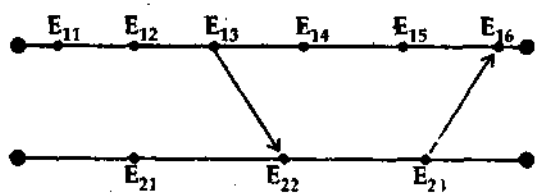
1. The network is reliable.
2. Latency is tending to zero.
3. Bandwidth is tending to infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

Q. 1(b). What are logical clocks? Why does a logical clock need to be implemented in distributed system? Explain with an example, what are the impacts of absence of global clock and shared memory.

Ans. A logical clock is a mechanism for capturing chronological and causal relationships in a distributed system. Logical clock algorithms of note are:

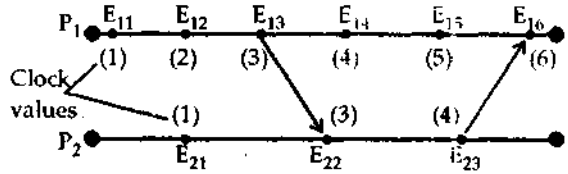
- Lamport timestamps, which are monotonically increasing software counters
- Vector clocks, that allow for total ordering of events in a distributed system.
- Matrix clocks, an extension of vector clocks that also contains information about other processes' views of the system.

Q. 1(c). Consider the following space time diagram for two processes P_1 and P_2



Obtain the Lamport time stamp for each event. List the events which casually affect the event E_{22}.

Ans. Lamport time stamp for



Clearly from above figure

$E_{13} \rightarrow E_{22}$, $E_{22} \rightarrow E_{23}$, $E_{23} \rightarrow E_{16}$

and therefore

$E_{22} \rightarrow E_{16}$

In other words, E22 causally affects event E_{16}.

Q.1.(d). What do you mean by Casual Ordering of messages? Discuss the salient features of Broadcast based protocol that make

the uses of Vector clock which ensures Causual Ordering of messages.

Ans. Causal ordering of messages: The purpose of causal ordering of messages is to insure that the same causal relationship for the "message send" events correspond with "message receive" events. (i.e. All the messages are processed in order that they were created.)

Part-ii Schiper-Eggli-Sandoz Protocol

Instead of maintaining a vector clock based on the number of messages sent to each process, the vector clock for this protocol can increment at any rate it would like to and has no additional meaning related to the number of messages currently outstanding.

Sending a message:

1. All messages are timestamped and sent out with a list of all the timestamps of messages sent to other processes.
2. Locally store the timestamp that the message was sent with.

Receiving a message:

- A message cannot be delivered if there is a message mentioned in the list of timestamps that predates this one.
- Otherwise, a message can be delivered, performing the following steps:
  1. Merge in the list of timestamps from the message:
     - Add knowledge of messages destined for other processes to our list of processes if we didn't know about any other messages destined for one already.
     - If the new list has a timestamp greater than one we already had stored, update our timestamp to match.
  2. Update the local logical clock.
  3. Check all the local buffered messages to see if they can now be delivered.

Q.1.(e). What do you mean by problem of mutual exclusion in Distributed System? What are the requirements of a good mutual exclusion algorithm? How does the performance of a Distributed algorithm?

Ans. Mutual Exclusion: Mutual exclusion (often abbreviated to mutex) algorithms are used in concurrent programming to avoid the simultaneous use of a common resource, such as a global variable, by pieces of computer code called critical sections.

Examples of such resources are fine-grained flags, counters or queues, used to communicate between code that runs concurrently, such as an application and its interrupt handlers. The problem is acute because a thread can be stopped or started at any time.

Part-ii & III

To illustrate: suppose a section of code is altering a piece of data over several program steps, when another thread, perhaps triggered by some unpredictable event, starts executing. If this second thread reads from the same piece of data, the data, which is in the process of being overwritten, is in an inconsistent and unpredictable state. If the second thread tries overwriting that data, the ensuing state will probably be unrecoverable. These critical sections of code accessing shared data must therefore be protected, so that other processes which read from or write to the chunk of data are excluded from running

A mutex is also a common name for a program object that negotiates mutual exclusion among threads, also called a lock.

Q.1.(f). What are the Token and Non-token based algorithm ? Explain Lamport's algorithm with example.

Ans. Token Vs nontoken based algorithm:

We use *tokens* to encapsulate anything that needs to be shared by the team, including information, tasks and resources. The tokens are efficiently routed through the team via the use of local decision theoretic models. Each token is used to improve the routing of other tokens leading to a dramatic performance improvement when the algorithms work together.

Nontoken based algorithm : A station which requests a Critical Section (CS) competes in order to be alone to use the unique channel dedicated to this CS. To reach this goal, we derive a Markov process which guarantees that each station will enter the CS. More precisely, we show that, in presence of collision detection n/ln2 broadcast

rounds are necessary in the average case to satisfy n (n unknown) stations wishing to enter the same CS.

Lamport's algorithm: Pseudocode

```
// declaration and initial values of global variables
   Entering: array [1..N] of bool = {false};
   Number: array [1..N] of integer = {0};
 1 lock(integer i) {
 2 Entering[i] = true;
 3 Number[i] = 1 + max(Number[1], ..., Number[N]);
 4 Entering[i] = false;
 5 for (j = 1; j <= N; j++) {
 6 // Wait until thread j receives its number:
 7 while (Entering[j]) { /* nothing */ }
 8 // Wait until all threads with smaller numbers or with the same
 9 // number, but with higher priority, finish their work:
10 while ((Number[j] != 0) && ((Number[j], j) < (Number[i], i))) {
11 /* nothing */
12 }
13 }
14 }
15
16 unlock(integer i) {
17 Number[i] = 0;
18 }
19
20 Thread(integer i) {
21 while (true) {
22 lock(i);
23 // The critical section goes here...
24 unlock(i);
25 // non-critical section...
26 }
27 }
```

Q. 2. Attempt any two parts of the following—

Q.2.(a). (i) Explain the deadlock handling strategies in *f* distributed system.

(ii). Explain the control organization for Distributed deadlock detection.

Ans. (i).Deadlock handling strategies in distributed system: Distributed deadlocks can occur in distributed systems when distributed transactions or concurrency control is being used. Distributed deadlocks can be detected either by constructing a global wait-for graph, from local wait-for graphs at a deadlock detector or by a distributed algorithm like edge chasing. *Phantom deadlocks* are deadlocks that are detected in a distributed system.

Often neither deadlock avoidance nor deadlock prevention may be used. Instead deadlock detection and process restart are used by employing an algorithm that tracks resource allocation and process states, and rolls back and restarts one or more of the processes in order to remove the deadlock. Detecting a deadlock that has already occurred is easily possible since the resources that each process has locked and/or currently requested are known to the resource scheduler or OS.

(ii). Control organization for distributed deadlock detection: edge-chasing is an algorithm for deadlock detection in distributed systems. Whenever a process *A* is blocked for some resource, a probe message is sent to all processes *A* may depend on. The probe message contains the process id of *A* along with the path that the message has followed through the distributed system. If a blocked process receives the probe it will update the path information and forward the probe to all the processes it depends on. Non-blocked processes may discard the probe.

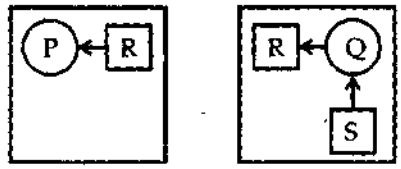
If eventually the probe returns to process *A*, there is a circular waiting loop of blocked processes, and a deadlock is detected. Efficiently detecting such cycles in the "wait-for graph" of blocked processes is an important implementation problem.

Q.2.(b). A centralized global deadlock detector holds the union of local wait-for graphs. Give an example to explain how a phantom deadlock could be detected if a waiting transaction in a deadlock cycle abort during the deadlock detection procedure.

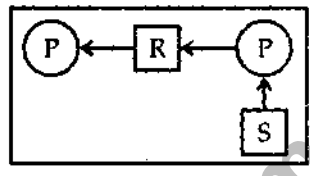
Ans. A Wait-For Graph is a directed graph used for deadlock detection in operating systems and relational database systems. a system that allows concurrent operation of multiple

processes and locking of resources and which does not provide mechanisms to avoid or prevent deadlock must support a mechanism to detect deadlocks and an algorithm for recovering from them.

when the wait-for graph is computed at a central location that collects messages from the various systems involved in transactions, it is easy to find also false, or spurious, or phantom deadlocks as in the following picture



Machine 1 Machine 2



Coordinator

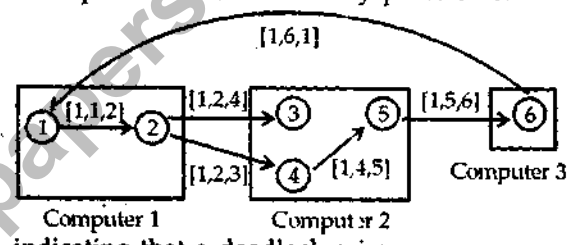
when Q first releases S and then P requests S, but the messages delivered to the coordinator that collects the messages arrive in the wrong order. Then the coordinator believes that a deadlock has arisen and kills unnecessarily P or Q. Of course, if Q was following a 2-phase locking policy, it would not release any lock until it had acquired every lock it needed. Thus phantom deadlocks would not arise if 2-phase locking is followed.

The Chandy-Misra-Haas algorithm detects deadlocks in a distributed system (resources with multiplicity 1) without requiring a coordinator. When process Pi requests a resource held by process Pj, it sends a message of form

[Requestor of resource,

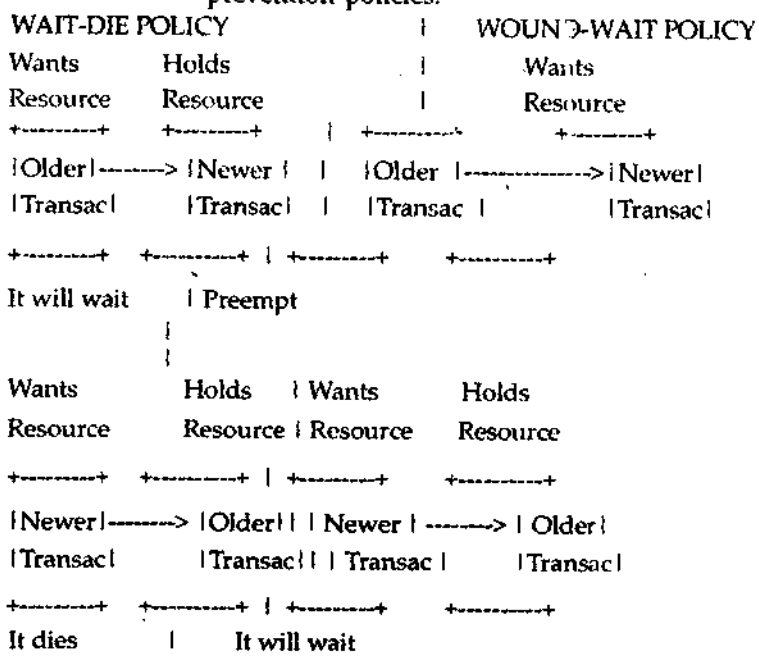
source of message,

destination of message] to Pj, thus in this case Pi sends to Pj the message [i, i, j]. In turn Pj will send to each process Ph such that Pj requests a resource held by process Ph, the message [i, j, h]. The same will be done by Ph, etc. if the original requestor Pi receives a message of form [i, *, i], then it will know that there is a loop, that a deadlock will arise, thus it will delay the request or abort. For example, here is what happens in the following system when process P1 requests a resource held by process P2:



indicating that a deadlock arises.

For distributed deadlock prevention can be used timestamps indicating the start of transactions and then ordering priority in case of conflictson the basis of the time stamp. The following diagram describes two deadlock prevention policies:



Q.2.(c)(i). What are the shortcomings of Ramamoorthy's two phase algorithm for deadlock detection?

(ii). Show that Byzantim agreement cannot always be reached among four processors if two processors are faulty.

Ans.(i). Shortcoming's of Ramamoorthy's two phase algorithm for dead lock detection:

- Each site maintains a status table of all processes initiated at that site: includes all resources locked & all resources being waited on.
- Controller requests (periodically) the status table from each site.
- Controller then constructs WFG from these tables, searches for cycle(s).
- If no cycles, no deadlocks.
- Otherwise, (cycle exists): Request for state tables again.
- Construct WFG based *only* on common transactions in the 2 tables.
- If the same cycle is detected again, system is in deadlock.
- Later proved: cycles in 2 consecutive reports *need not* result in a deadlock. Hence, this algorithm detects false deadlocks.

(ii). A Byzantine fault is an arbitrary fault that occurs during the execution of an algorithm by a distributed system. It encompasses those faults that are commonly referred to as "crash failures" and "send and omission failures". When a Byzantine failure has occurred, the system may respond in any unpredictable way, unless it is designed to have Byzantine fault tolerance.

These arbitrary failures may be loosely categorized as follows:

- a failure to take another step in the algorithm, also known as a crash failure;
- a failure to correctly execute a step of the algorithm; and
- arbitrary execution of a step other than the one indicated by the algorithm.

For example, if the output of one function is the input of another, then small round-off errors in the first function can produce much larger errors in the second. If the second function were fed into a third, the problem could grow even larger, until the values produced are worthless. Another example is in compiling source code. One minor syntactical error early on in the code can produce large numbers of perceived errors later, as the compiler gets out-of-phase with the lexical and syntactic information in the source program.

Steps are taken by processes, the abstractions that execute the algorithms. A faulty process is one that at some point exhibits one of the above failures. A process that is not faulty is correct.

The Byzantine failure assumption models real-world environments in which computers and networks may behave in unexpected ways due to hardware failures, network congestion and disconnection, as well as malicious attacks. Byzantine failure-tolerant algorithms must cope with such failures and still satisfy the specifications of the problems they are designed to solve. Such algorithms are commonly characterized by their resilience t, the number of faulty processes with which an algorithm can cope.

Many classic agreement problems, such as the Byzantine Generals' Problem, have no solution unless $t < n / 3$, where n is the number of processes in the system.

Q. 3. Attempt any two of the following:

Q.3.(a). What are the communication models proposed for the distributed objects? Explain the concept of remote method invocation with a suitable example.

Ans. Communication models for distributed objects:

Distributed objects are implemented in Objective-C using the Cocoa API with the NSConnection class and supporting objects.

- Distributed objects are used in Java RMI.
- CORBA lets one build distributed mixed object systems.
- DCOM is a framework for distributed objects on the Microsoft platform.
- DDObjects is a framework for distributed objects using Borland Delphi.
- JavaSpaces is a Sun specification for a distributed, shared memory (spaces based)

- Pyro is a framework for distributed objects using the Python programminglanguage.
- Distributed Ruby (DRb) is a framework for distributed objects using the Ruby programming language.

Part-ii

RMI (Remote Method Invocation) is a way that a programmer, using the Java programming language and development environment, can write object-oriented programming in which objects on different computers can interact in a distributed network. RMI is the Java version of what is generally known as a remote procedure call (RPC), but with the ability to pass one or more objects along with the request. The object can include information that will change the service that is performed in the remote computer. Sun Microsystems, the inventors of Java, calls this "moving behavior." For example, when a user at a remote computer fills out an expense account, the Java program interacting with the user could communicate, using RMI, with a Java program in another computer that always had the latest policy about expense reporting. In reply, that program would send back an object and associated method information that would enable the remote computer program to screen the user's expense account data in a way that was consistent with the latest policy. The user and the company both would save time by catching mistakes early. Whenever the company policy changed, it would require a change to a program in only one computer.

RMI is implemented as three layers:

- A stub program in the client side of the client/server relationship, and a corresponding skeleton at the server end. The stub appears to the calling program to be the program being called for a service. (Sun uses the term *proxy* as a synonym for stub.)
- A Remote Reference Layer that can behave differently depending on the parameters passed by the calling program. For example, this layer can determine whether the request is to call a single remote service or multiple remote programs as in a multicast.
- A Transport Connection Layer, which sets up and manages the request.
- A single request travels down through the layers on one computer and up through the layers at the other end.

Q.3.(b). Discuss how a public key scheme can be used to solve the key distribution problem in a private key cryptographic scheme.

Ans. Key Distribution Scheme in Private key distribution: It uses Diffie-Hellmar key exchange (D-H) is a cryptographic protocol that allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher.

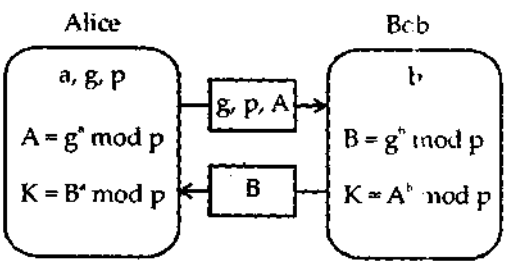
Synonyms of Diffie-Hellman key exchange include:

- Diffie-Hellman key agreement
- Diffie-Hellman key establishment
- Diffie-Hellman key negotiation
- Exponential key exchange

Hellman suggested the algorithm be called Diffie-Hellman-Merkle key exchange to the invention of public-key cryptography.

Although Diffie-Hellman key agreement itself is an *anonymous* (non-*authenticated*) key-agreement protocol, it provides the basis for a variety of authenticated protocols, and is used to provide perfect forward secrecy in Transport Layer Security's ephemeral modes.

Description



$K = A^b \bmod p = (g^a \bmod p)^b \bmod p = g^{ab} \bmod p$
$= (g^b \bmod p)^a \bmod p = B^a \bmod p$

Diffie-Hellman key exchange: The simplest, and original, implementation of the protocol uses the Multiplicative group of integers

modulo p, where p is prime and g is primitive root mod p.

Q.3.(c). 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 ?

Ans. Feature of AFS design make more scalable than NFS:

The main strengths of AFS are its:

+ caching facility
+ security features
+ simplicity of addressing
+ scalability
+ communications protocol

Part-ii Scalability: With location independence comes scalability. An architectural goal of the AFS designers was client/server ratios of 200:1 which has been successfully exceeded at some sites.

Improved security: Firstly, AFS makes use of Kerberos to authenticate users. Secondly, AFS uses access control lists (ACLs) to enable users to restrict access to their own directories.

Replicated AFS volumes: AFS files are stored in structures called Volumes. These volumes reside on the disks of the AFS file server machines. Volumes containing frequently accessed data can be read-only replicated on several servers.

Improved robustness to server crash: The Cache Manager maintains local copies of remotely accessed files. If a fileserver crashes, the client's locally cached file copies remain readable but updates to cached files fail while the server is down.

"Easy to use" networking: Accessing remote file resources via the network becomes much simpler when using AFS. Users have much less to worry about: want to move a file from a remote site? Just copy it to a different part of /afs.

Communications protocol: AFS communications protocol is optimized for Wide Area Networks. Retransmitting only the single bad packet in a batch of packets and allowing the number of unacknowledged packets to be higher (than in other protocols.

Improved system management capability: Systems administrators are able to make configuration changes from any client in the AFS cell (it is not necessary to login to a fileserver). With AFS it is simple to effect changes without having to take systems off-line

Q. 4. Attempt any two of the following—

Q.4.(a). The two-phase commit protocol is a centralized protocol where the decision to abort or commit is taken by the co-ordinator. Design a decentralized two-phase commit protocol where no site is designated to be a co-ordinator.

Ans. Centralized Two phase commit protocol:

Two-phase commit is a standard protocol in distributed transactions for achieving ACID properties. Each transaction has a coordinator who initiates and coordinates the transaction.

In the two-phase commit the coordinator sends a prepare message to all participants (nodes) and waits for their answers. The coordinator then sends their answers to all other sites. Every participant waits for these answers from the coordinator before committing to or aborting the transaction. If committing, the coordinator records this into a log and sends a commit message to all participants. If for any reason a participant aborts the process, the coordinator sends a rollback message and the transaction is undone using the log file created earlier. The advantages of this are all participants reach a decision consistently, yet independently.

However, the two-phase commit protocol also has limitations in that it is a blocking protocol . For example, participants will block resource processes while waiting for a message from the coordinator. If for any reason this fails, the participant will continue to wait and may never resolve its transaction. Therefore the resource could be blocked indefinitely. On the other hand, a coordinator will also block resources while waiting for replies from participants. In this case, a coordinator can also block indefinitely if no acknowledgement is received from the participant

To reduce blocking, we propose a backup

commit (BC) protocol by attaching multiple backup sites to the coordinator site. In this protocol, after receiving responses from all participants in the first phase, the coordinator communicates the final decision to the backup sites in the backup phase. Afterwards, it sends the final decision to the participants. When blocking occurs due to the failure of the coordinator site, the participant sites can terminate the transaction by consulting a backup site of the coordinator. In this way, the BC protocol achieves non-blocking property in most of the coordinator site failures.

The BC protocol suits best for World Wide Web (or Internet) environments where a server has to face high rush of electronic commerce transactions that involve multiple participants. Also in the Internet environment, sites fail frequently and messages take longer delivery time. In this situation with extra hardware, the BC protocol reduces the blocking problem without involving expensive communication cycle as compared to 3PC. Through simulation experiments it has been shown that the BC protocol exhibits superior throughput and response time performance over the 3PC protocol and performs closely with the 2PC protocol.

Q.4.(b). Describe how a non-recoverable situation could arise if write locks are released after the last operation of a transaction but before its commitment

Ans. Non Recoverable situation arise if write locks are released: The present invention provides a system that supports recovery in the event a previous process holding a lock used for mutual exclusion purposes loses ownership of the lock. This loss of ownership may occur due to the previous process dying or the lock becoming unmapped. Under the present invention a process first attempts to acquire the lock. If the attempt to acquire the lock returns with an error indicating that the previous process holding the lock lost ownership of the lock, the process attempts to make program state protected by the lock consistent. If the attempt to make the program state consistent is successful, the system reinitializes and unlocks the lock. Otherwise, the system marks the lock as unrecoverable so that subsequent processes attempting to acquire the lock are notified that the lock is not recoverable. One aspect of the present invention includes receiving a notification in an operating system that a process died, and determining if the process died while holding a lock. If the process died while holding the lock, the system marks the lock to indicate to subsequent acquirers of the lock that a previous holder of the lock died, and unlocks the lock so that other processes may acquire the lock. According to one aspect of the present invention, if the attempt to acquire the lock returns with an error indicating the lock is not recoverable, the process performs operations to work around the program state that is inconsistent, and reinitializes the lock.

Write ahead logging (WAL) is a family of techniques for providing atomicity and durability (two of the ACID properties) in database systems. In a system using WAL, all modifications are written to a log before they are applied. Usually both redo and undo information is stored in the log. The purpose of this can be illustrated by an example. Imagine a program that is in the middle of performing some operation when the machine it is running on loses power. Upon restart, that program might well need to know whether the operation it was performing succeeded, half-succeeded, or failed. If a write-ahead log were used, the program could check this log and compare what it was supposed to be doing when it unexpectedly lost power to what was actually done. Based on this comparison, the program could decide to undo what it had started, complete what it had started, or keep things as they are.

Q.4.(c). Explain how the two-phase commit protocol for nested transaction ensures that if the top-level transaction commits all the right descendents are committed or aborted.

Ans. The two-phase commit protocol (2PC) is a distributed algorithm that lets all nodes in a distributed system agree to commit a