UNIT-4

SYNCHRONIZATION AND REPLICATION

1) CLOCKS, EVENTS AND PROCESS STATES

* Process communicate only by msgs. Each process p_i can be viewed as to execute a sequence of actions that fall in one of the following categories:

a) Sending a msg

b) Receiving a msg

c) Performing a computation that alters its state S_i.

* **Physical clock**

→ computer physical clocks are electronic devices that count oscillations occurring in a crystal at a definite frequency. This count can be stored in a counter register.

126

* The clock o/p can be read by s/w and scaled into a suitable time unit. This value can be used to timestamp any event experienced.

* consider two clocks A and B, where clock B runs slightly faster than clock A by appx two seconds per hour. This is the clock drift of B relative to A. At one point in time, the difference in time b/w the two clocks is appx 4 seconds. This is the clock skew at that particular time.

* Successive events will correspond to different timestamps only if the clock resolution is smaller than the rate at which events can occur.

* The rate at which events occur

depends on such factors as the length of the processor instruction cycle.

* Working:

1. Oscillation at a well-defined frequency
2. Each crystal oscillation decrements the counter by 1
3. When counter gets 0, its value reloaded from the holding register.
4. When the counter is 0, an interrupt is generated, which is called a clock tick.
5. At each clock tick, an interrupt service procedure add 1 to time stored in m/y.

2) SYNCHRONIZING PHYSICAL CLOCK

* clock synchronization is done by two method:
  a) internal synchronization
  b) external synchronization

127

* Synchronization in a synchronous s/m

→ A synchronous distributed s/m is one in which the following bounds are defined:

1. The time to execute each step of a process has known lower and upper bounds.
2. Each msg transmitted over a channel is received within a known bounded time.
3. Each process has a local clock whose drift rate from real time has a known bound.

* cristian's method for synchronizing clocks

→ christian's algm is centralized passive time server type algm

→ The simplest algm for setting the

time would be to simply issue a RPC to a time server and obtain the time.

→ cristian's algm suffers from the pblm that afflicts all single-server algms: the server might fail and clock-synchronization will be unavailable. It is also subject to malicious interference.

* Berkeley algm

→ Time server is a active m/c

→ The server polls each m/c periodically, asking it for the time. The time at each m/c may be estimated by using cristian's mtd to account for n/w delays

→ when all the results are in, the master computes the average time

→ instead of sending the updated time back to the slaves, which would introduce further uncertainty due to n/w delays,

it sends each m/c the offset by which its clock needs adjustment.

* Network Time protocol

→ Cristian's method and the berkeley algm are intended for intranets.

* Goals:

1. Enable clients across the internet to be accurately synchronized to UTC despite msg delays.
2. Provide a reliable service that can survive lengthy losses of connectivity.

* NTP servers synchronize with one another in one of three modes in the order of increasing accuracy.

1. Multicast or high speed local LANs
2. Procedure call mode
3. Symmetric mode

* NTP servers synchronize modes are as follows:

1. Multicast
2. Procedure - call
3. Symmetric - mode.

3) LOGICAL TIME & LOGICAL CLOCKS

* For a certain class of algms, it is the internal consistency of the clocks that matters. The convention in these algorithm is to speak of logical clocks.

* Event ordering

→ A person with one watch knows what time it is. But a person with two or more watches is never sure

→ Lamport defined a relation called happens before, represented by →

→ The relation → on a set of events of a s/m is the relation satisfying three conditions:

192

1. If a and b are events in the same process and a comes before b, then $a \rightarrow b$.

2. If a is the sending event of a msg by one process, and b is the receipt event of msg, then $a \rightarrow b$.

3. If $a \rightarrow b$, $b \rightarrow c$ then $a \rightarrow c$

* Lamport Timestamp

→ Lamport algm is used to synchronize the logical clock. It uses happens-before relationship to provide a partial ordering of events:

1) All processes use a counter (clock) with initial value of zero.

2) The counter is incremented by and assigned to each event, as its timestamp

3) A send (msg) event carries its timestamp

4) For a receive (msg) event the counter

is updated by

max (receive - counter, message - timestamp) + 1

→ Requirements:

a) if $a \rightarrow b$ then $T(a) < T(b)$

b) $T(a) \neq T(b)$ for any two different events a and b.

* Vector Timestamp

→ with Lamport's clocks, one cannot directly compare the timestamp of two events to determine their precedence relationship

→ The main pblm is that a simple integer clock cannot order both events within a process and events in different processes.

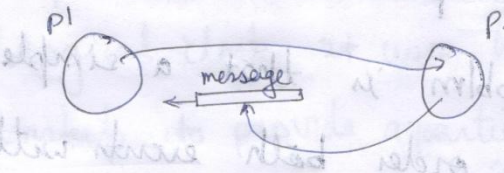
→ A vector clock s/m is a mechanism that associates timestamps with events such that comparing two events timestamps indicates whether those events are causally related.

134

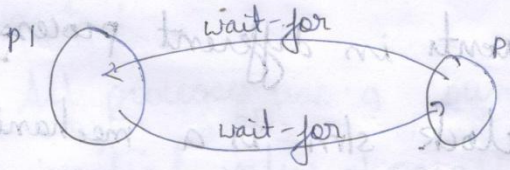
4) GLOBAL STATES

The global state of a distributed computation is the set of local states of all individual processes ~~introduced~~ involved in the computation plus the state of the communication channels.

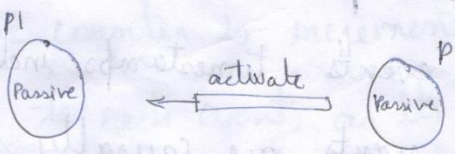
* Requirements of global state



a) Garbage collection



b) Deadlock



c) Termination

1. Distributed garbage collection
2. Distributed deadlock detection
3. Distributed termination detection
4. Distributed debugging

* GLOBAL STATES AND CONSISTENT CUTS

→ The global state of a distributed s/m is a collection of the local states of the processes and the channels.

→ OS cannot know the current state of all process in the distributed s/m. A process can only know the current state of all processes on the local s/m. Remote processes only know state information that is received by msgs

→ Need for global states: Many pbms in distributed computing can be cast as executing some action on reaching a particular state.

136

channel: Exists b/w two processes if they exchange msgs

state: Sequence of msgs that have been sent and received along channels incident with the process

snapshot: Records the state of a process

Distributed snapshot: A collection of snapshot one for each process.

* Global state

→ The global state can be defined as the union of all the snapshots.

→ To have a global consistent state then process state has registered a received message, the sending process state must own this message.

→ Useful to adapt any centralized algm over a distributed s/m

* Cut:

→ A cut is a set of cut events, one per node, each of which captures the state of the node on which it occurs.

* Chandy - Lamport Algm

→ chandy - lamport algm records a set of process and channel states such that the combination is a consistent global state. Communication channels assumed to be FIFO.

* Algorithm:

1. initiator process P_0 records its state locally.

2. Marker sending rule for process P_i:

After P_i has recorded its state, for each outgoing channel C_{hij}, P_i sends one marker message over C_{hij}

139

3. Marker receiving rule for Process P_i

Process P_i on receipt of a marker over channel C_{hji}.

If (P_i has not yet recorded its state) it

Records its process state now;

Records the state of C_{hji} as empty set;

starts recording messages arriving over other incoming channels;

else (P_i has already recorded its state)

P_i records the state of C_{hji} as the set of all messages it has received over C_{hji} since it saved its state.

5) COORDINATION & AGREEMENT

* Fundamental issue in OS is for a set of processes, how to co-ordinate their actions or to agree on one or more values.

*. Further issue is how to consider and deal with failures when designing algms.

* Distributed mutual exclusion is needed when processes access a resources or collection of resources and we require their updates to be consistent. Algorithms to achieve distributed mutual exclusion must be evaluated on the basis of their efficiency and their behaviours under failure conditions.

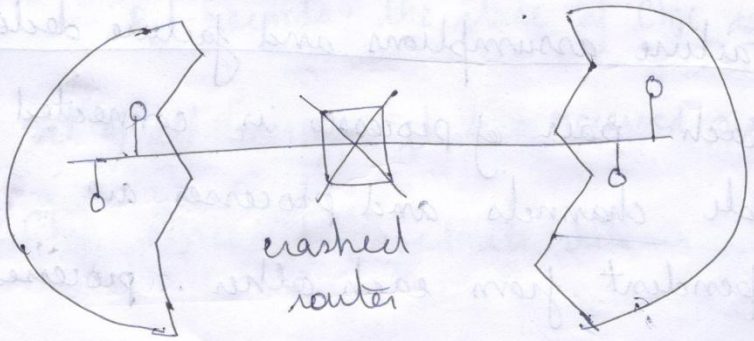
* Failure assumptions and failure detectors

→ Each pair of processes is connected by reliable channels and processes are independent from each other. processes only fail by crashing

140

→ underlying n/w components may suffer failures, but reliable protocols recover. Reliable channel eventually delivers msg. No bound as in an asynchronous s/m

→ A collection of processes is split into several groups that cannot communicate. Failure of router b/w two n/ws may mean that a collection of four processes is split into 2 pairs. Here intra-pair communication is possible but because of router fail this communication is also not possible. This is called as n/w partition.



n/w partition

→ Failure detector is object/code in a process that detects failures of other processes. Failure detector is not every-time accurate. This is category as unreliable failure ~~detector~~ detector. unreliable failure ~~detector~~ detector is one of two values:

a) unsuspected
b) suspected

1) Unsuspected: Failure is unlikely, ex: failure detector has recently received communication from unsuspected peer. This may be inaccurate

2) Suspected: Indication that peer process failed. ex: No msg received in quite sometime Also this may be inaccurate because peer process hasn't failed, but the communication link is down, or peer process is much slower than expected.

142

6) DISTRIBUTED MUTUAL EXCLUSION

* Mutual exclusion ensures that concurrent processes make a serialized access to shared resources or data. It requires that the actions performed by a user on a shared resources must be atomic

* In a distributed s/m ~~it~~ neither shared variable nor a local kernel can be used in order to implement mutual exclusion. Thus, mutual exclusion has to be based exclusively on msg passing, in the context of unpredictable msg delays and no complete knowledge of the state of the s/m.

* Mutual exclusion: Makes sure that concurrent proces shared resources or data

in a serialized way. If a process, say P_i is executing in its critical section, then no other processes can be executing in their critical sections

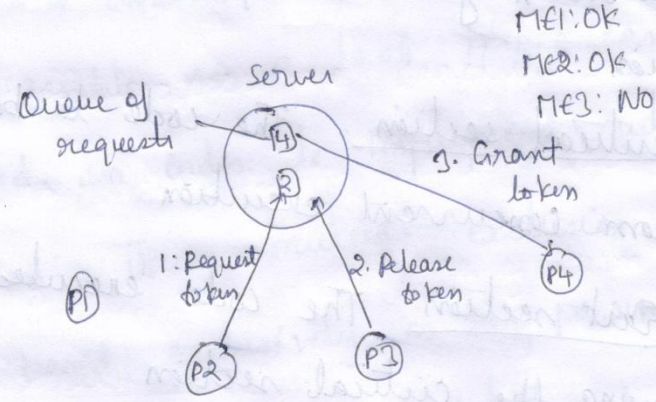
* Mutual exclusion is the fundamental issue in the design of distributed s/m's.
* Entry section: The code executed in preparation for entering the critical section
* Critical section: The code to be protected from concurrent execution
* Exit section: The code executed upon leaving the critical section
* Remainder section: The rest of the code.
* Performance metrics for mutual exclusion algm
  1. Message complexity
  2. Synchronization delay

144

3. Response time
4. System throughput

* Central Server algm

The simplest way to ensure mutual exclusion is through the use of a centralized server. Here server grants the permission to enter the critical section



Centralized server algorithm

* There is a conceptual token; process must be in possession of the token in order to execute the critical section.

* Ring - Based Algorithm

→ A simple way to arrange for mutual exclusion without the need for a master process, is to arrange the processes in a logical ring. The ring may of course bear little resemblance to the physical n/w or even the direct links b/w processes.

* If it requires access to the critical section, the process:

1. Retains the token
2. Performs the critical section and then
3. To relinquish access to the critical section
4. Forwards the token on to the next neighbour in the ring

Performance:

→ constant bandwidth consumption

→ Entry delay b/w 0 and N msg transmission times

146

8) ELECTIONS

* Many distributed algorithms require one process to act as co-ordinator, initiator or otherwise perform some special role. It does not matter which process take on this special responsibility, but one of them has to do it.

* If all processes are exactly the same, with no distinguishing characteristics, there is no way to select one of them to be special.

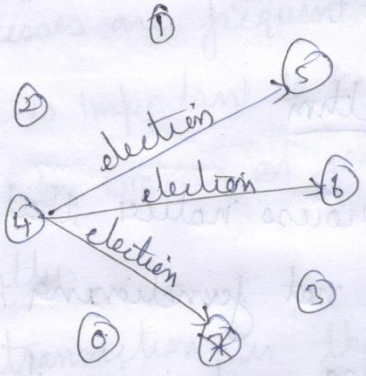
* In general, election algms attempt to locate the process with the highest process number and designate it as co-ordinator.

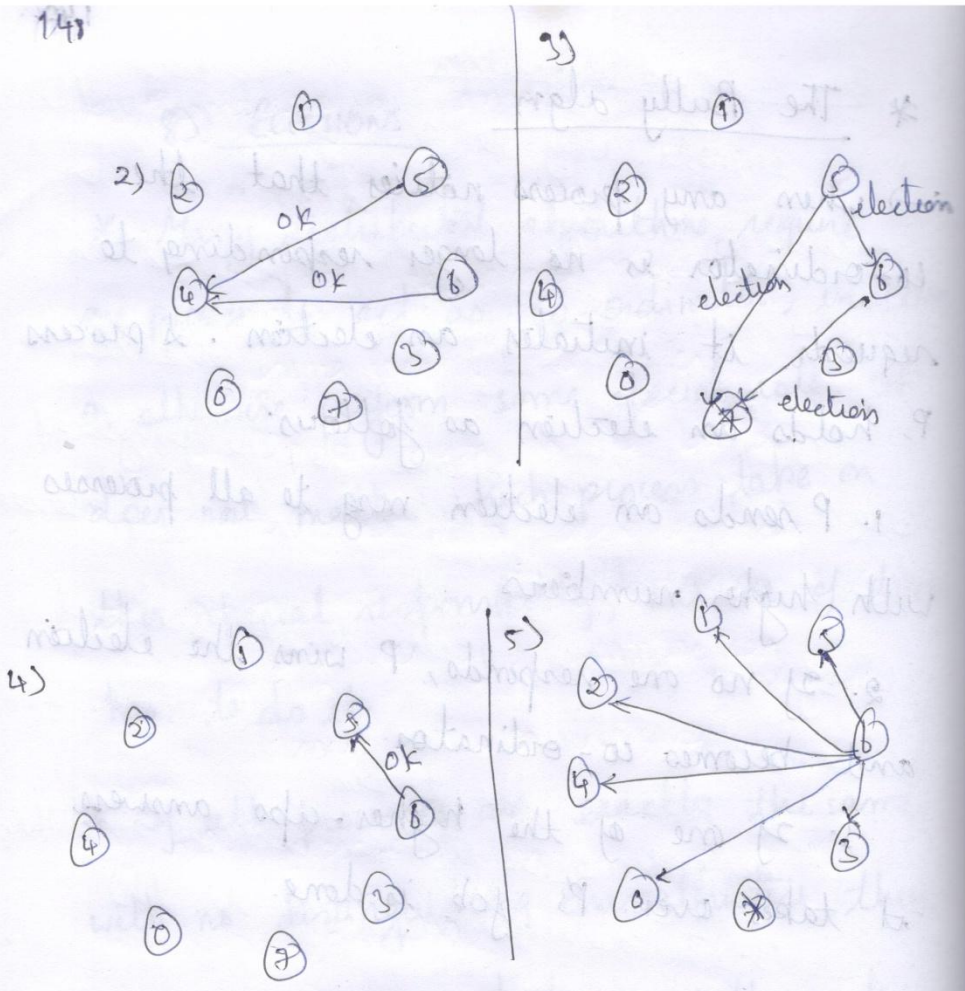
* The Bully algm

→ when any process notices that the co-ordinator is no longer responding to requests, it initiates an election. A process P, holds an election as follows:

1. P sends an election msg to all processes with higher numbers.
2. If no one responds, P wins the election and becomes co-ordinator.
3. If one of the higher-ups answers it takes over. P's job is done.

1)





* A Ring Algorithm

→ when any process notices that the co-ordinator is not functioning, it builds an ELECTION msg containing its own process number and sends the msg to its successor

→ If the successor is down, the sender skips over the successor and goes to the next member along the ring, or the one after that until a running process is located.

In each step, the sender adds its own process number to the list in the msg effectively making itself a candidate to be elected as co-ordinator.

8) TRANSACTIONS AND CONCURRENCY CONTROL

* Concurrent execution of user pgms is essential for good DBMS performance. Because disk accesses are frequent, and relatively slow, it is important to keep the CPU humming by working on several user pgms concurrently.

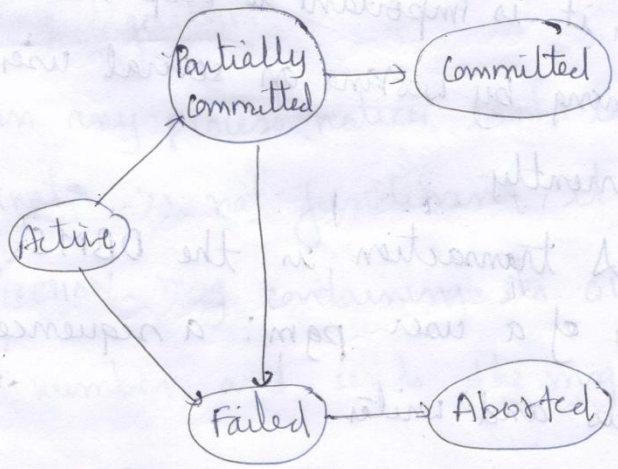
* A transaction is the DBMS's abstract view of a user pgm: a sequence of reads and writes.

153

* Examples of a transaction
  1. updating a record
  2. Locate the record on disk
  3. Bring record into buffer
  4. Update data in the buffer
  5. writing data back to disk

* Acid properties of a transaction
  1. Atomic - All or Nothing
  2. Consistent
  3. Isolation
  4. Durability

*



* Flat Transactions:

→ Flat transactions represent the simplest type of transaction, and for almost all existing s/ms, it is the only one that is supported at the application programming level.

→ A flat transaction is the basic building block for organizing an appln. into atomic actions. It can contain an arbitrary no. of simple actions: these actions may, in turn, be either protected, real or even unprotected, if necessary.

* Limitation

1) All or nothing is not always appropriate.

2) It is beneficial to commit partial results.

152

3) Trip planning: commit part of a transaction.

4) Bulk updates: can be expensive to undo all updates

* Concurrency control

→ Co-ordination of simultaneous transaction execution in a multiprocessing database s/m.

→ Ensure transaction serializability in a multi-user database

→ Lack of concurrency control can create data integrity and consistency pblms.

1. Lost updates
2. Uncommitted data
3. Inconsistent retrievals

a) NESTED TRANSACTION

* Nested transaction extends the transaction model by allowing transactions to be composed of other transactions. The outermost transaction in a set of nested transactions is called the top-level transaction

* Transactions other than the top-level transaction are called sub-transaction

The rules for committing of nested transactions are:

1) A transaction may commit or abort only after its child transactions have completed.

2) When a sub-transaction completes, it makes an independent decision either to commit provisionally or to abort. Its decision to abort is final.

154

3) When a parent aborts, all of its sub-transactions are aborted.

4) When a sub-transaction aborts, the parent can decide whether to abort or not.

5) If the top-level transaction commits, then all of the sub-transactions that have provisionally committed can commit too, provided that none of their ancestors has aborted.

* Advantages of nested transactions:

1. Subtransactions at one level may run concurrently with other subtransactions at the same level in the hierarchy.

2. Subtransactions can commit or abort independently.

155

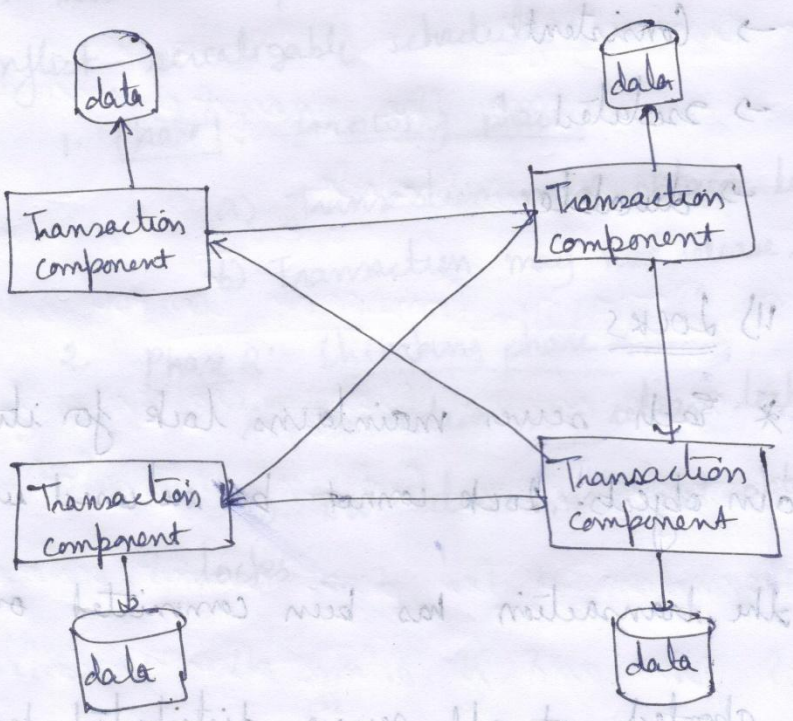
* Rules for nested transaction

1. A transaction may commit or abort only after its child transactions have completed

2. When a parent aborts, all of its subtransactions are aborted.

3. When a subtransaction aborts, the parent can decide whether to abort or not.

10) DISTRIBUTED TRANSACTION



* A distributed transactions more than one server involves.

* A distributed transaction is composed of several sub-transactions, each running on a different site. Each database Manager (DM) can decide to abort

* Distributed transactions are hard

→ Atomic
→ Consistent
→ Isolated
→ durable

ii) Locks

* Each server maintains lock for its own objects. Lock cannot be released until the transaction has been committed or aborted at all servers. distributed deadlock

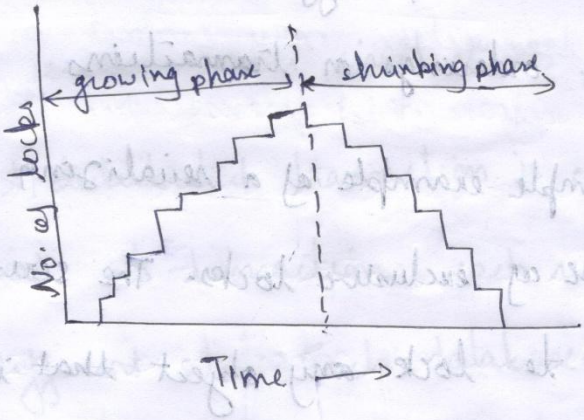
might occur if different servers impose different orderings on transactions.

* A simple example of a serializing mechanism is the use of exclusive locks. The server attempts to lock any object that is about to be used by any operation on a client's transaction.

* Two-Phase locking protocol

→ This is a protocol which ensures conflict-serializable schedules.

1. Phase 1: Growing phase
   a) Transaction may obtain locks
   b) Transaction may not release locks
2. Phase 2: Shrinking phase
   a) Transaction may release locks.
   b) Transaction may not obtain locks.



* Advantage -
guaranteeing serializability without having to check the schedules themselves.

* Disadvantage -
limiting the amount of concurrency, deadlock and livelock.

12) OPTIMISTIC CONCURRENCY CONTROL

* If we assume that, validation is fast in single server. But in distributed transaction of two phase commit protocol, it takes and it delays other transactions

from entering validation until a decision on the current has been obtained

* In distributed optimistic concurrency control, each server applies a parallel validation protocol. If parallel validation is used, transaction will not suffer from commitment deadlock.

* Timestamp ordering

→ Timestamps are used to order transactions according to their starting times. In distributed transactions, we require that each co-ordinator issue globally unique timstamp

→ In distributed transaction, all servers are jointly responsible for ensuring that the operation in serially equivalent manner. even if local clock is not synchronized, then also same ordering of transaction can be achieved.

160

13) Atomic Commit Protocols

* Atomicity principle requires that either all the distributed operations of a transaction complete or all abort. At some stage, client executes close Transaction(). Now, atomicity requires that either all participates or the co-ordinator commit or all abort.

* One-phase commit protocol

→ The co-ordinator tells the participant whether to commit or abort.

→ This does not allow one of the servers to decide to abort - it may have discovered a deadlock or it may have crashed and been restarted.

→ Advantage:

1. Simple protocol fewer overheads

2. Low latency due to fewer disks writes

→ disadvantage

1. It can only handle immediate consistency constraints as there is no voting phase

2. It can not handle the deferred consistency constraints

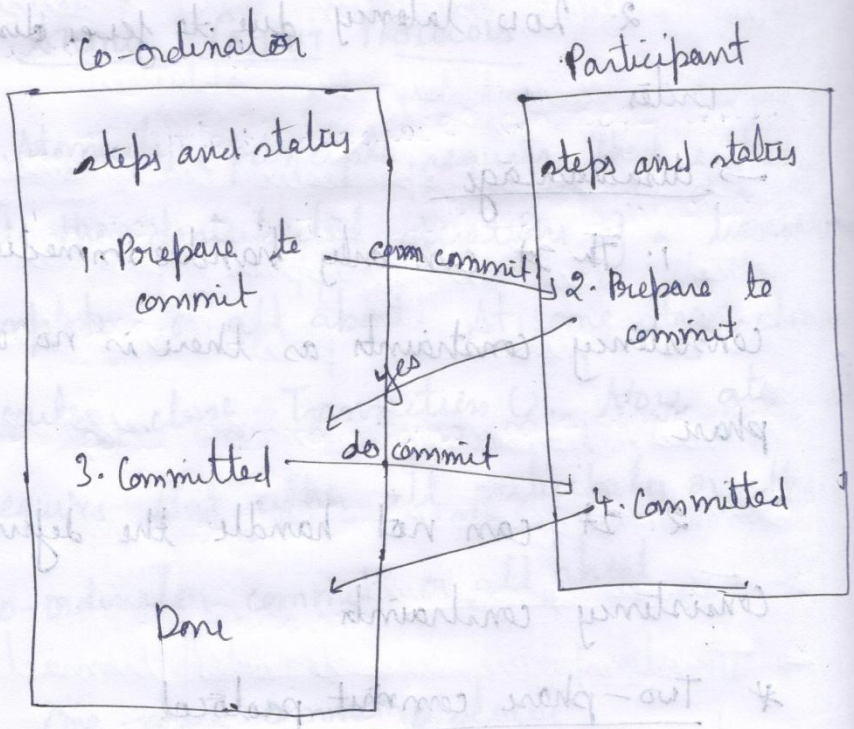
* Two-phase commit protocol

→ 2PC is the std protocol for making commit and abort atomic. It is designed to allow any participant to choose to abort a transaction. If one part of the transaction is aborted, then the whole transaction must also be aborted.

→ Phase 1: preparation

→ phase 2: The final COMMIT

162.



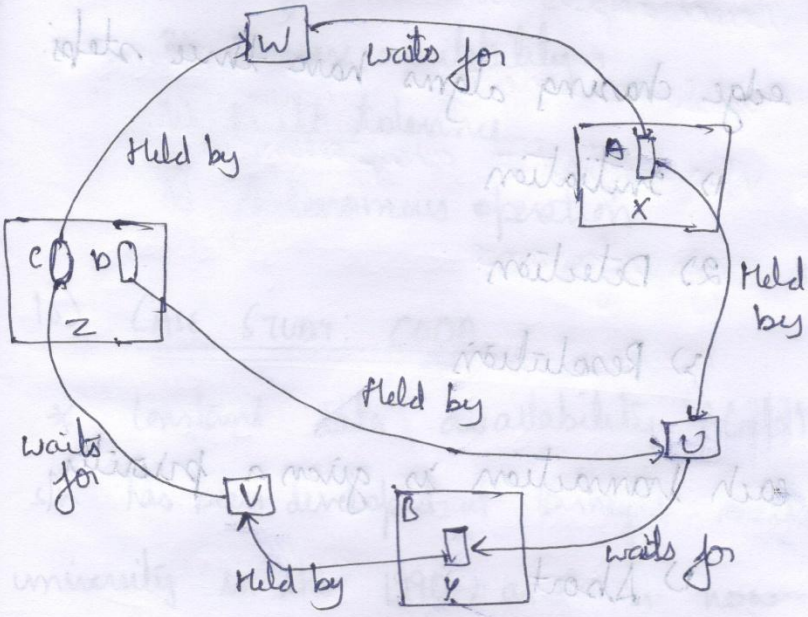
Communication in 2 Phase commit Protocol

14) DISTRIBUTED DEADLOCKS

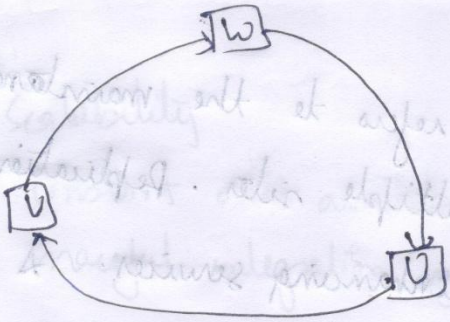
* A state in which each member of a group of transactions is waiting for some other members to release a lock

* The following fig shows the wait for graph. So the global wait-for-graph

is held in part by each of the several servers involved, communication b/w these servers is required to find cycles on the graph.



a) distributed deadlock



b) wait-for graph.

164

* Edge chasing

→ To detect deadlock an edge chasing techniques or path pushing techniques is used.

→ edge chasing algms have three steps:

1) Initiation
2) Detection
3) Resolution

→ each transaction is given a priority:

1) Abort
2) Initiation
3) Forward

8) REPLICATION

* Replication refers to the maintenance of copies at multiple sites. Replication is a technique for enhancing services. A logical object is implemented by a collection of

physical copies called replicas

* The goal is to improve a service's performance, to increase its availability, or to make it fault tolerant

1) Performance
2) Increase availability
3) Fault tolerance
4) Autonomous operation

16) CASE STUDY: CODA

* Constant data availability (Coda) file s/m has been developed at Carnegie mellon university in the 1990, and is now integrated with a no. of popular UNIX based operating s/m.

* goals:

1. Scalability
2. Constant data availability
3) Graceful integration of the use of the file s/m.

162

* Features:

1. disconnected operation for mobile computing
2. Is freely available under a liberal license
3. sever application
4. Good scalability
5. n/w bandwidth adaptation
6. High performance