2017 Fall

1a) Distributed System and its design challenges.

Simply, Ds is a collection of logically related data that is distributed over different processing nodes of computer network.

A distributed system is a collection of independent computers that appear to the users of the system as a single computer.

The design challenges of Distributed system are:

1. Heterogeneity: heterogenous component that must be able to ~~operate~~ interoperate, apply to all of the following: network, os, hardware. It mask the difference in components.
   Eg: Middleware provide heterogeneity.

2. Openness: Ds must be extensible. openness (feature add) determines whether the system can be extended and re-implemented in various ways. ie can changing architecture works as a new system.

3. Reliability: Ds should be more reliable than a single system. Availability, redundancy and fault tolerence mares system more reliable.

4. Performance: Ds should have most performance. performance loss due to communication delays and fault tolorent.

5. Security: Resources should access to authorized user and used in the way they are intended

7. Scalability: Scalable DS operate effectively and efficiently at many different scales, ranging from a small intranet to internet. Scalable when no. of ~~increase~~ resource and no. of users increases.

8. Failure Handling: Failures in DS are partial, that is some components fail while others continue to function.

9. Concurrency: With Concurrency, services and applications can be shared by clients in a DS.

10. Transparency: Hiding of the separation of component in a distributed systems from the user and the application programmer.

11b. What is IDL (Interface Definition Language)? What are the components of a CORBA environment.

IDL is a generic term for a language that lets a program or object written in one programming language communicate with another program written in another language. In Distributed technology, its important that new objects be able to sent to any platform environment and discover how to run in that environment. An Object Request Broker (ORB) is an example of a program that would ~~be~~ use an IDL to broker communication between one object program and another one.

- An Interface specifies the API that ~~can~~ the clients can use to invoke operation on objects.

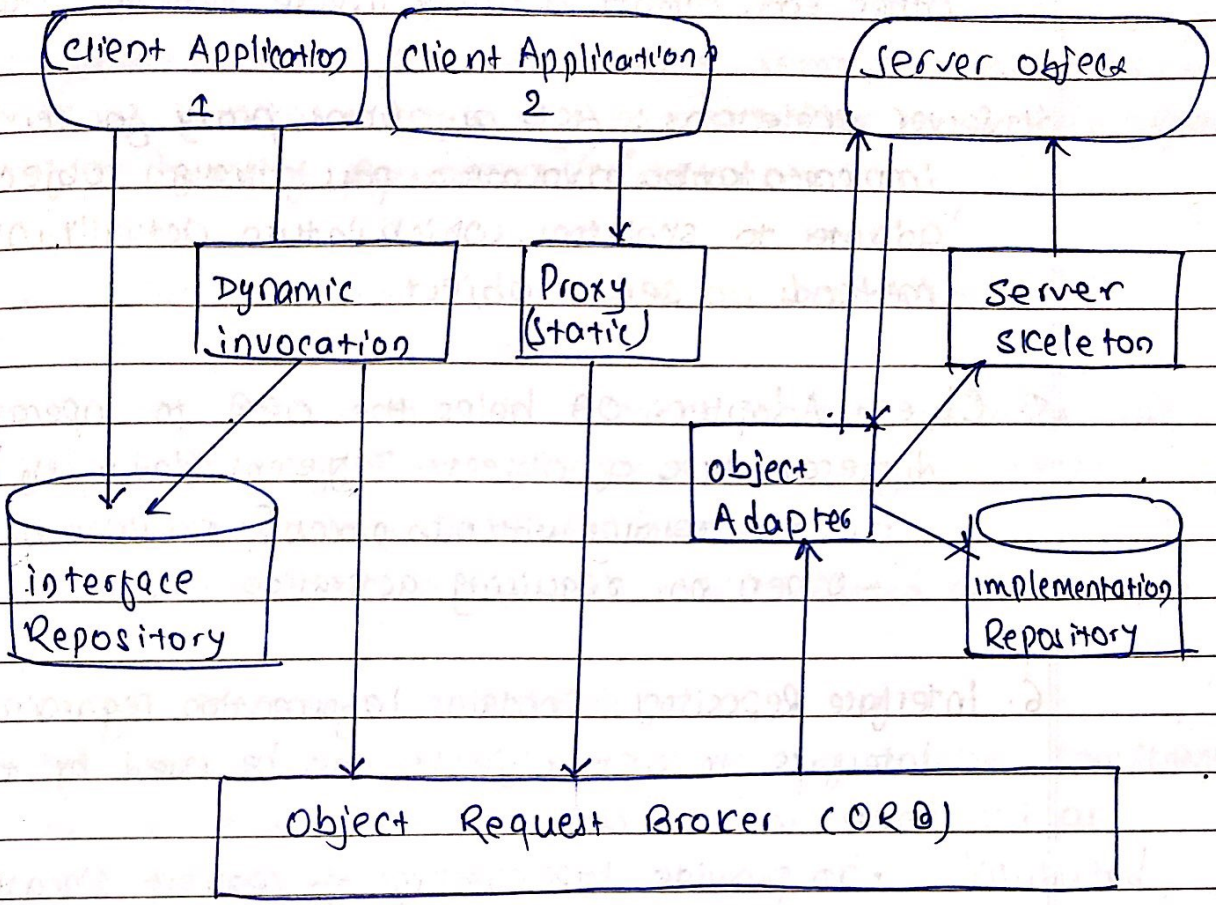


fig: CORBA environment / Architecture

1. ORB: Gives the Comm^n infrastructure that is capable of relaying object requests across distributed platforms. client calls the object implementation through interfaces provided by ORB.
   - separates client and server implementation

2. IDL: CORBA uses IDL for defining interfaces between client and server. ORB vendors provide specific IDL compilers for supported languages - create target language stubs and skeletons for building CORBA clients and servers. Eg: C, C++, Smalltalk, Java

3. Client Stubs: client side proxy for server. It joins the client at one end and to the ORB core at the

other end. clients actually invoke methods on stub objects.

4. Server skeletons: Acts as client proxy for server implementation. Invocation pass through object adapter to skeletons, which in turn actually invoke methods on server object.

5. Object Adapter: OA helps the ORB to operate with different type of objects. Different kind of obj implementations
   - objects residing with own process & requiring activation.
   - others not requiring activation

6. interface Repository: Contains information regarding the interface to ORB objects. can be used by the ORB in 2 ways:
   - To provide typechecking of request signatures, whether a request was issued through DII or stub
   - To check correctness of inheritance Graph.

- client objects can use it:
  - To manage installation and distribution of interface definitions around your network.
  - Language compilers may use them to generate stubs and skeletons.
- can be shared by more than ORB or one ORB may refer to more than one interface repository.

7. Implementation Repository: Contains all the information regarding object implementation
- Provides a persistent record of how to activate and invoke operation on object implementation.
- CORBA gives vendors free-hand in handling implementation

2|a]

8. Dynamic invocation interface:
- Generic interface for making remote invocations.
- Uses interfaces repository at run-time to discover interfaces
- No need of pre-compiled stubs.

What are the programming models for Distributed Application? Explain them
- objects that can receive remote methods invocations are remote invocation are called remote objects and they implements a remote interface.
The programming models for Distributed applications are

1. Local procedure call (LPC):
- LPC occurs when a component invokes another component in the same namespace. LPC is of course, as old as modular programming and usually assumed in distributed settings. We explicitly include LPC because programmers employ it in DS wherever possible because of its inherent efficiency.

2. Remote Procedure call (RPC):
- Client calls a procedure implemented and executing on a remote computer
- Call as if it was a local procedure. The procedure must already reside on computation target.
- If necessary, a stub that handles parameter marshalling is sent to the invoking namespace. Example: JAVA RMI

Remote Method Invocation (RMI):
- Local object invokes methods of an object residing on a remote Computer

3. Code on demand (COD):

COD denotes a local computation that requires a remote component, which is downloaded to the local namespace. Example: Java applets

4. Remote Evaluation (REV)

REV occurs when a client desires the remote execution of a local component. P moves component c to the desired namespace B where the computation occurs as shown in fig(c).

5. Mobile Agent (MA):

MA describes a component that can move itself, while it is executing, from one namespace to another.

models	current Location	Target	moves component
REV	local	remote	yes
RPC	remote	remote	no
COD	remote	local	yes
MA	remote	remote	yes
LPC	local	local	no

fig: Distributed programming models parameterized

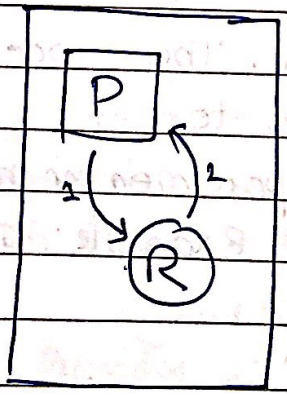


fig: a Local procedure call (LPC)

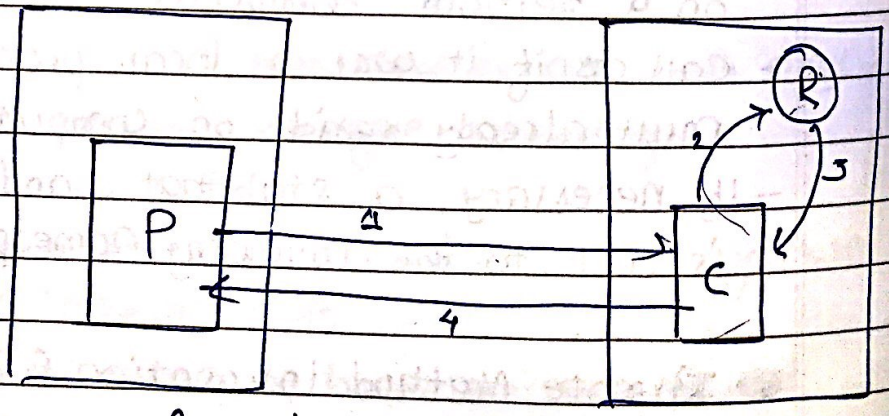


fig: b. Remote procedure call

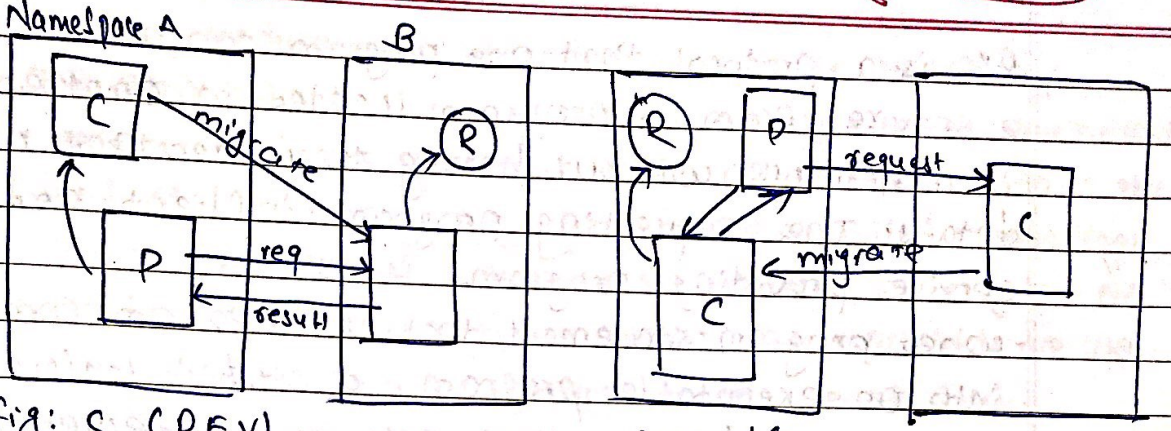


fig: c (REV)

fig: d (COD)

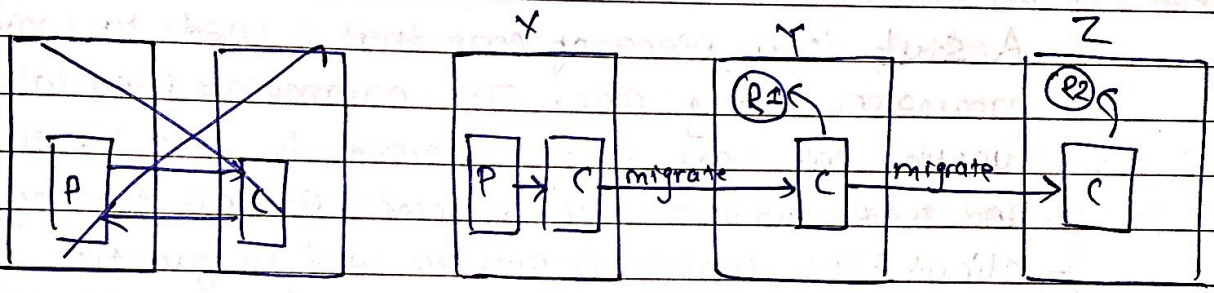


fig: Mobile Agent (MA)

[2|b] What is RPC? Explain the role of client & server stub procedures in RPC with diagrams.

→ A remote procedure call is similar to Remote method invocation (RMI). A client program calls a procedure in another program running in a server process. RPC is implemented over a request-response protocol

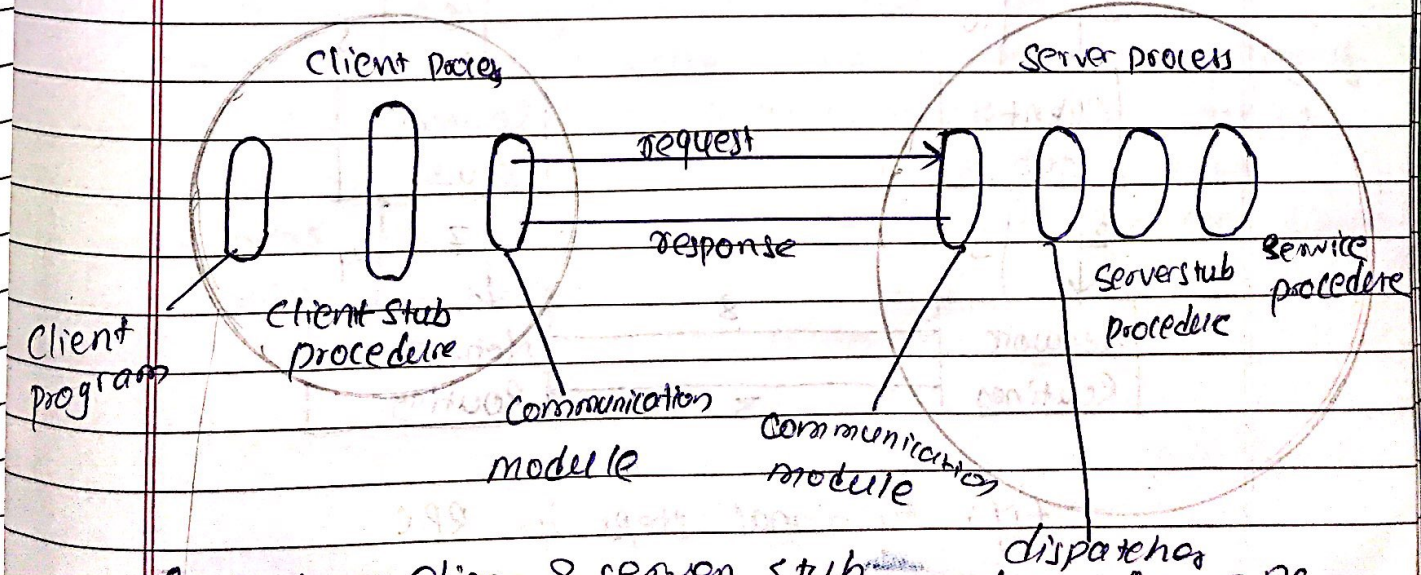


fig: role of client & server stub procedures in RPC.

RPC is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details. The requesting program is client and the service providing programs is server.

- when program statement that use RPC are compiled into an executable program, a stub is included in the compiled code that acts as the representative of RPC.
- A stub is a piece of code that is used to convert parameters during RPC. The parameters used in a function call have to be converted because the client and server computers use different address (directory) spaces.
- client stub convert parameters used in function calls and reconvert the result obtained from the server after function execution
- Server stub reconverts the parameter passed by client and convert the results back after function execution.

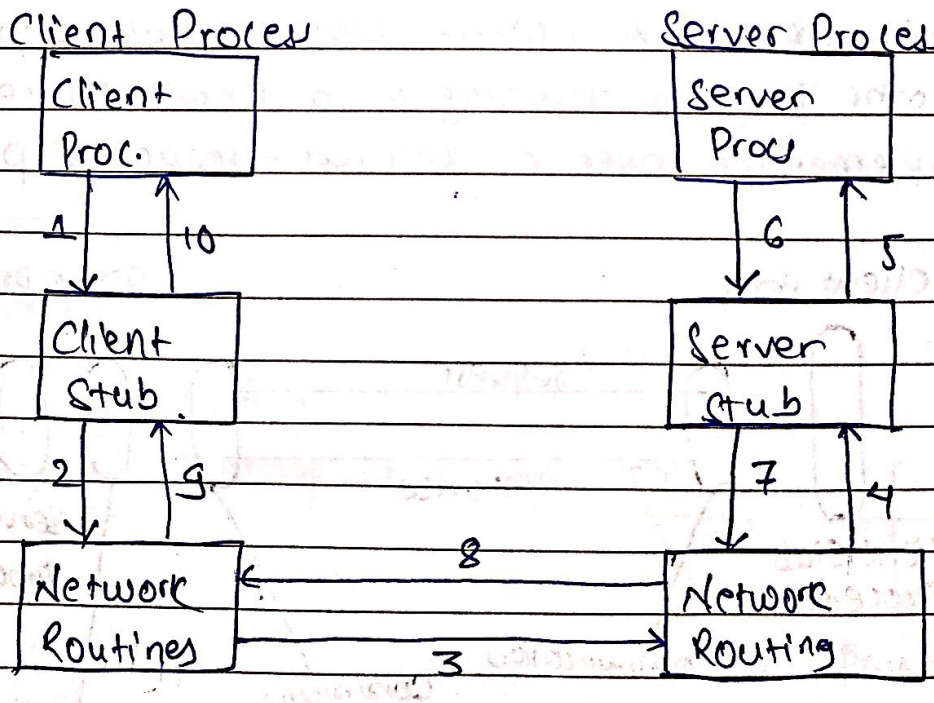


fig: fuctional steps in RPC

1. Client procedures calls the client stub in the normal way.
2. The client stub builds a message including ~~message~~ parameters, name or number of procedure to be called etc and calls the local OS. The packaging of argumen in a nlw called marshalling.
3. The client's OS sends the msg to remote OS via a system call to the local kernel. To transfer the message, some protocols are used.
4. The remote OS gives the message to the server stub.
5. The server stub unpacks the parameters and call the server
6. Server does the work and returns the result to the server stub.
7. The server stub packs a message and call its local OS.
8. The server's OS sends the message to client's OS.
9. The client's OS gives the message to the client stub.
10. The stub unpacks the result and returns to the waiting client procedure.

[3|a] What is clock synchronization. Explain the network time protocol for physical clock synchronization

→ Clock synchronization is the process of setting all the cooperating systems of distributed network to the same logical or physical clock

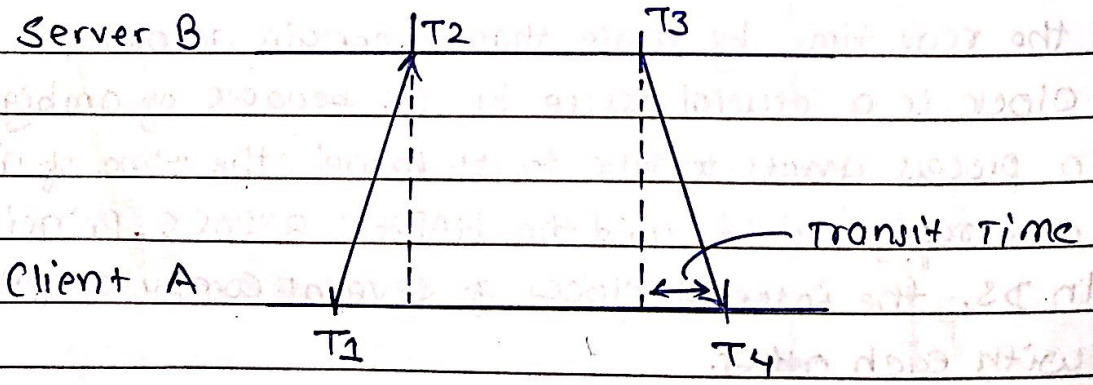
- logical clocks: To provide consistent event ordering
- Physical clocks: clocks whose values must not deviate from the real time by more than a certain amount
  - clock is a crucial issue in Ds because of ambigiuity. If a process wants to talk to its kernel, the time of asking and reply should hold the HAPPEN BEFORE principle
- In DS, the internal clocks of several computers may differ with each other.

Network Time protocol (NTP)

- NTP is the most commonly used internet time protocol and the one provides best accuracy. Computers often includes NTP software in OS. The client software periodically gets updated from one or more servers (If, average them)
- Time servers listen to NTP request on port 123, and reply a UDP/IP data packets in NTP format, which is a 64-bit timestamp in UTC seconds since Jan 1, 1900 with a resolution of 200 pico-second. Many NTP client software for PC gets time from single server (No-averaging). The client is called simple NTP (SNTP), a simple version of NTP.
- Enable clients across internet to be accurately synchronized to UTC despite message delays. It provides reliable service

NTP Synchronization Modes:

- Multicast: Server periodically multicasts its time to its clients in the subnet (low accuracy)
- Remote Procedure Calle: server responds to client request with its actual timestamp (medium accuracy)
- Symmetric mode: used to synchronize between time servers (peer-to-peer), (high accuracy)



- A request time of B at its own time T_1
- B receives request at its T_2, records
- B responds at its T_3, sending value T_2 and T_3
- A receives response at its T_4

- A knows $(T_4 - T_1)$ from its own clock.
- B reports T_3 and T_2 in response to NTP request
- A computes total transit time of
$(T_4 - T_1) - (T_3 - T_2)$

- One way transit time is approx. 1/2 total ie.
$$\frac{(T_4 - T_1) - (T_3 - T_2)}{2}$$

- B's clock at T_4 reads approximately

$$T_3 + \frac{(T_4 - T_1) - (T_3 - T_2)}{2} = \frac{(T_4 - T_1) + (T_3 + T_2)}{2}$$

- Thus, difference between B and A clock at T_4 is

$$\frac{(T_4 - T_1) + (T_3 + T_2)}{2} - T_4 = \frac{(T_2 - T_1) + (T_3 - T_3)}{2}$$

[3(b)] Explain Lamport's timestamp algorithm for logical clock synchronization

→ - In a DS, it is not possible in practice to synchronize time across entities (processes) within the system; hence process can use the concept of logical clock based on the events through which they communicate. When two entities ~~donot exchange~~ communicate by message passing, then the send event is said to be 'happen before' the receive event, and logical order can be established.

- Lamport invented a simple mechanism by which the happened-before ordering can be captured numerically. A Lamport Logical clock is a incrementing software counter maintained in each process.

Algorithm:

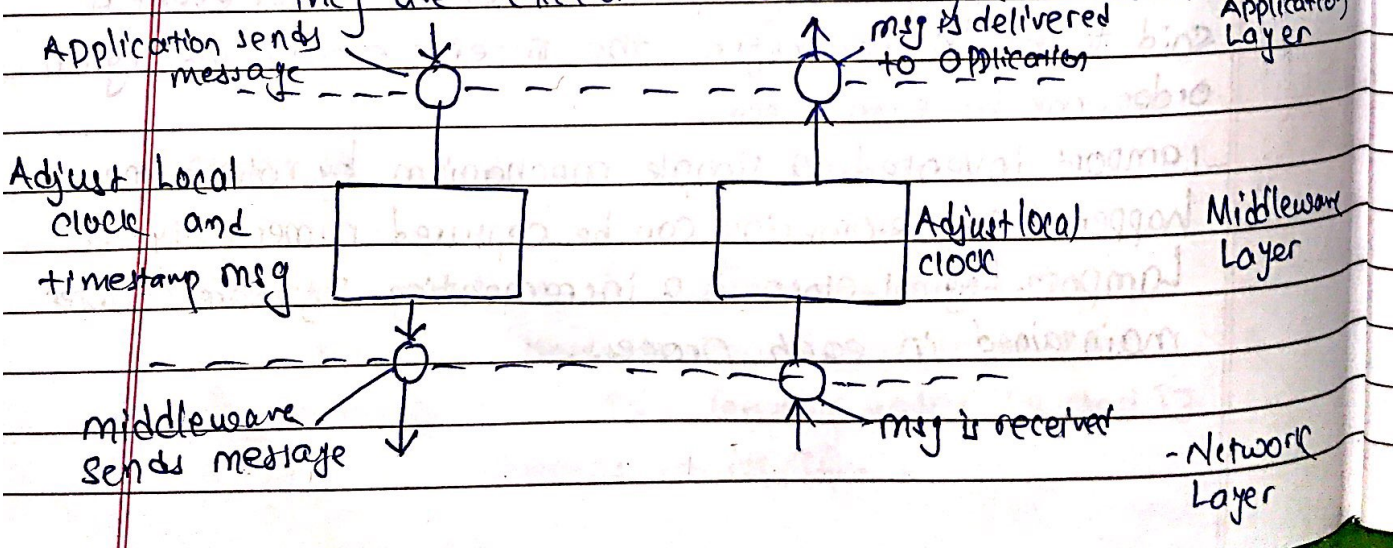
1. A process increments its counter before each event in that process.
2. when process sends a message; it includes its counter value with the message.
3. On receiving a message, the receiver process set its counter to be greater than the maximum of its own value and the received value before it considers the message received.

- conceptualy, this logical clock can be thought of as a clock that only has meaning in relation to message moving between process. when a process receives a message; it resynchronize its local clock with that sender.

-The happened-before relation is a partial ordering of events in Ds such that:

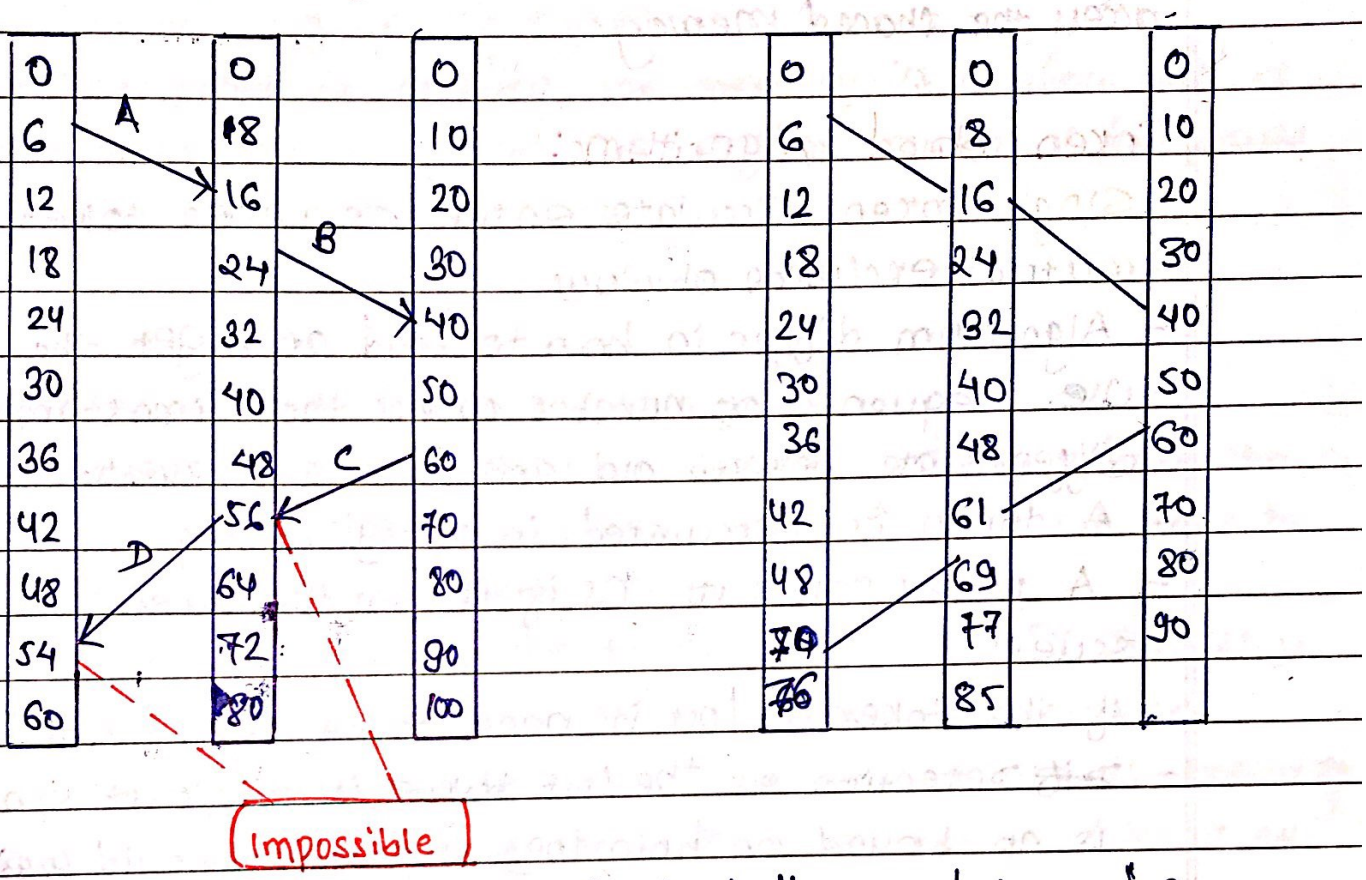
1. If A and B are events in same process, A was executed before B, then $A \Rightarrow B$
2. If A is the event of sending a msg by one process and B is the event of receiving that by another process then $A \Rightarrow B$
3. If $A \Rightarrow B$ and $B \Rightarrow C$ then $A \Rightarrow C$

• If two events are not related by $\Rightarrow$ relation, then they are executed concurrently (no casual relationship)



Example: Lamport's Algorithm

- Three process each with its own local clock. The clock run at different rates.
- Lamports algorithm correct the clock



Note: ts(A) < ts(B) doesnot imply A 'happened before" B

Consideration: The logical clock be set so that there is minimum of one clock 'tick' (increment of the counter) between events and;

Mutual Exclusion: Token based algorithm to obtain mutual exclusion.

A mutual exclusion (mutex) is a program object that prevent simultaneous access to shared resource. This concept is used in concurrent programming with a critical section, a piece of code in which process or threads access a shared resource.

- If two processes are allowed to concurrently be in competing

critical sections, then incorrect results may be computed called a Race condition. The process of ensuring that this destructive interaction doesnot occur is called mutal exclusion. It makes sure that only one process shall acess the shared memory.

Token Based Algorithms:
- Single token circulates, enter cs when token is present
- mutual exclusion obvious
- Algorithm differ in how to find and get the token.
- Use sequence of number rather than timestamps to differentiate between old and current requests.
- A token is circulated in a logical ring
- A process enters its CS if it has the token.

Issues:
- If the token is lost, it needs to be regenerated
- ~~Dif~~ Detection of the lost token is difficult since there is no bound on how long a process should wait for the token.
- If a process can fail, it needs to be detected and then by passed.
- when noone wants to enter, processes keep on exchanging messages to circulate the token

A. Token Ring algorithms:
- A simple way to solve mutual exclusion. Token is circulated among all nodes sequentially, a node which has token has right to acess the server.
- A logical ring is calculated in which each process is assigned in a position in the ring, initially a process is given the token. The token circulates around the ring.

- when a process acquires the token from its neighbour, it checks to see if it is attempting to enter a critical region. If so ~~the~~, the processor enter the region, finishes it work and leaves the region. Then it passes the token along the ring. in clockwise
- If the process is handled the token by its neighbour and is not interested in entering a critical region, it just passes it along the ring.

~~B. Suzuki~~

B. Suzuki Kasami Algorithm:

- If a site wants to enter the CS and it doesnot have the token, it broadcasts a REQUEST message for the token to ~~all the token~~ to all other sites.
- A site which possesses the token sends it the requesting site upon the receipt of its REQUEST message
- If a site receive a REQUEST message when it is ~~erecting~~ executing the CS, it sends the token only after it has completed the execution of the CS.
- This algorith uses sequence numbers instead.

4[b] Chang and Roberts (Ring algorithm) election algorithm

- Chang and Roberts is a ring-based election algorithm used to find a process with the largest identification.
- The algorithm works for any number of processes N, and doesnot require any process to know how many proce are in the ring
- Assumes that the links are unidirectional (clockwise)

Algorithm

1. Intially each process in the ring is marked as non-partici
2. A process that notices a lack of leader starts an elect It creates an election message containing its UID. It then

sends this msg clockwise to its neighbour.

3. Everytime a process sends or forwards an election message, the process also marks itself as a participant.

4. When a process receives an election message it compares the Unique Identification Number (UID) in the message with its own UID.

a. If the UID in the election message is larger, the process unconditionally forwards the election message in a clockwise direction.

b. If the UID in the election message is smaller, and the process is not yet a participant, the process replaces the UID in the message with its own UID, sends the updated election message in a clockwise direction.

c. If the UID in the election message is smaller and process is already a participant, the process discards the election message

Note: participant process is the process has already sent out an election message with a UID at least as large as its own UID.

d. d. If the UID in the incoming election message is same as the UID of the process that process ~~starts to~~ starts acting as the leader

5. When process starts acting as a leader, it begins the second stage of the algorithm

a. The leader process marks itself as non-participant and sends an elected message to its neighbour announcing its election and UID

b. When a process receives an elected message, it marks itself as non-participant, records the elected UID, and forwards the elected message unchanged.

c. When the elected message reaches the newly elected leader, the leader discards that message, and the election is over

This algorithm is for ring networks. Each message goes in the network from one process to another, ie no broadcasting - its neighbour. This could be imagined as Linked List.

5[a] What is the reason for Replication in Distributed System. Explain active and passive replication technique for fault tolerence in DS.

→ Data Replication is a common technique in DS. There are two reasons for data replication:

- it increases the reliability of a system - If one replica is unavailable or crashes, use another and it protect against corrupted data.
- It improves the performance of the system
  - Scale with size of the DS (replicated web servers).
  - Scale in geographically DS (web proxies)

- The key issue is need to maintain consistency of replicated data:
  - If one copy is modified, others become inconsistent.

Replication Technique for Fault Tolerance (Service provision technique):

- Active Replication
- Passive Replication

Both technique provide the fault tolerance. In passive replication there is only one server (called primary) that processes client request. In active replication each client request is processed by all the replicas:

A. Passive Replication (Primary Backup):

There is only one server (called primary) that process client requests. After processing a request, the primary server updates the state on the other (backup) servers and

sends back the response to the client. If the primary server fails, one of the backup server takes place. Passive replication may be used even for non-deterministic process. The disadvantage of passive replication compared to active is that in case of failure the response is delayed.

In active replication, each client request is processed by all the ~~prote~~ servers. This requires that the processes hosted by the servers is deterministic. Deterministic means that, given the same intial state and a request ~~sequen~~ sequence, all processes will produce the same response sequence and end up in the same final state. Inorder to make all the servers receive ~~a message or none, plus that they all receive messages in the same order~~ the same sequence of operations, an atomic broadcast protocol must be used. An atomic broadcast protocol guarantees that either all the server receive a message or none, plus that they all receive message in same order.
disadvantage: most real world servers are non-deterministics. ~~but~~ still active replication is preferable choice when dealing with RTS that require quick response under the presence of faults or with systems that must handle byzantine faults.

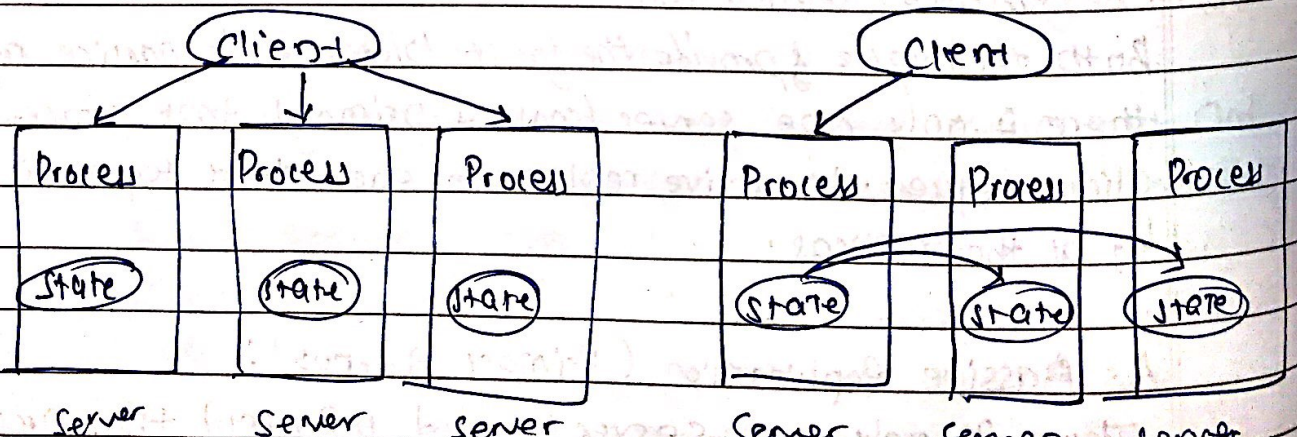


fig: Active Replication

fig: Passive Replication.

5[b] What are the approaches to fault tolerant? Explain two-phase and three-phase commit protocol

The system can provide services even in the presence of faults, such systems are called fault Tolorent. There are two approaches to fault Tolorent:

a. Mask failures: System continues to provide its specified function(s) in the presence of failures. eg: voting protocols.
b. Well-defined failure behaviour:
- system exhibits a well define behaviour in the presence of failures.
- It may or it may not perform its specified function(s), but facilitates actions suitable for fault recovery.
- Example: commit protocols.

Distributed commit: Either all members of a group decide to perform an operation or none of them perform the operation.
is done by using atomic Commitment protocole (ACP):

A. One Phase Commit Protocol:
- The scheme is not fault Tolerent.
- One site is designed as a coordinator which tells all the other processes whether or not to locally perform the operation in question
- But what if process cannot perform the operation? There is no way to tell the
- Solutions: The two Phase and three-phase commit.

B. Two Phase Commit Protocol: (blocking Commit protocol)
A feature of Transaction processing that enables databases to be ~~functioned~~ returned to the pre-transaction state if some

error condition occur. It ensures either all the db are update or none of them, so db are synchronized.

- One site is elected coordinator of the Transaction T. and others are participants.

Coordinator:

Phase 1: When coordinator is ready to commit the Transaction
- Place Prepare(T) state in log on stable storage
- Send Vote_request(T) message to all participants
- wait for replies.

phase 2: Coordinator
- If any participant replies Abort(T)
  - place Abort(T) in log
  - Send Global_Abort(T) message to all participants
  - Locally abort transaction T
- If all participants reply Ready_to_Commit(T)
  - Place Commit(T) in log
  - Send Global_Commit(T) message to all participants
  - proceed to commit transaction locally

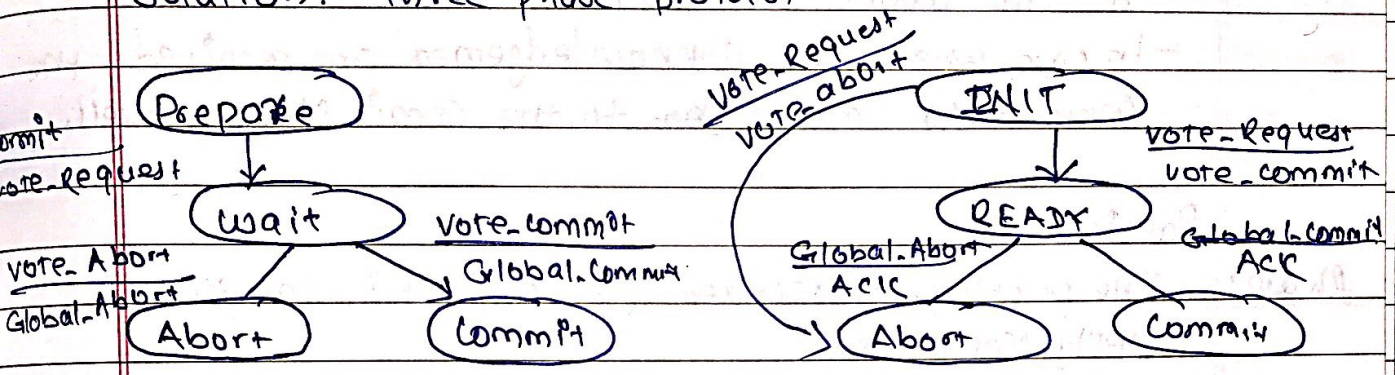
Participants

Phase 1: participants get Vote_Request(T) from Coordinator
- place Abort(T) or Ready(T) state in local log.
- Reply with Abort(T) or Ready_To_Commit(T) msg to coordi
- If Abort(T) state, locally abort Transaction.

phase 2: participant
- wait for Global_Abort(T) or Global_Commit(T) msg from coordinator
- Place Abort(T) or Commit(T) state in logical log.
- Abort or Commit locally per msg.

Problems: It can lead to both coordinator and the group member blocking, which may lead to the dreaded deadlock.
- If Coordinator crashes, group member may block until coordinator recovers.

Solution: Three phase protocol



C. Three Phase Commit Protocol: (Non-blocking Commit):
- There is no state from which a transition can be made to either Commit or Abort.
- There is no state where it is not possible to make a final decision and from which transaction can be made to commit.

Coordinator
- Phase 1: Coordinator receives a transaction Request,
  - If there is a failure, aborts the transaction
  - else coordinator sends a CanCommit? message to the participants ~~to the~~ and ~~wait~~ moves to the waiting state
- Phase 2:
  - If there is a failure, timeout or if the Coordinator receives a 'No' msg in waiting state, Coordinator abort the transaction and sends abort msg to all the participants
  - else coordinator receive yes msg with time, it sends Precommit message to all participants and moves to the prepare state.

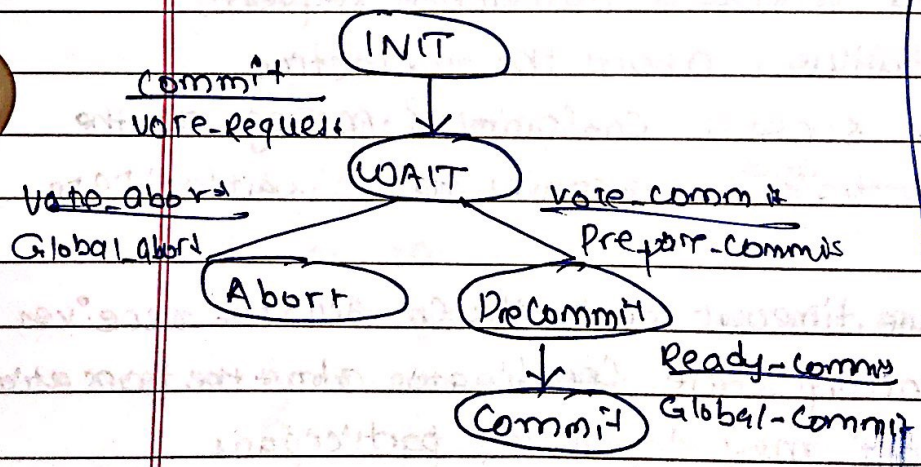
Phase 3: If the coordinator succeeds in prepared state, it will move to the commit state
- However if the coordinator times out while waiting for an acknowledgement from a participant - It will abort the transaction
- In case where all acknowledgement are received, the coordinator moves to the commit state as well.

Participants

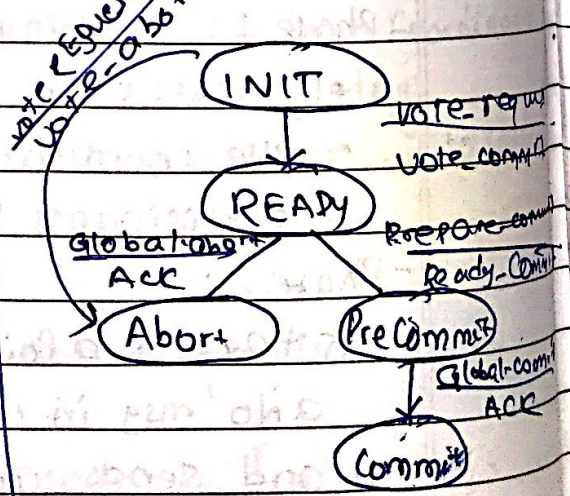
Phase 1: The participants receives a ~~con~~commit ? participant reply Yes or No
- If there is a failure it moves to the abort state.

Phase 2: In prepared state, if participants receive abort msg, fails or timeout, it aborts.
If it receives precommit msg, it sends an Ack msg back and await ~~a~~ final commit or abort.

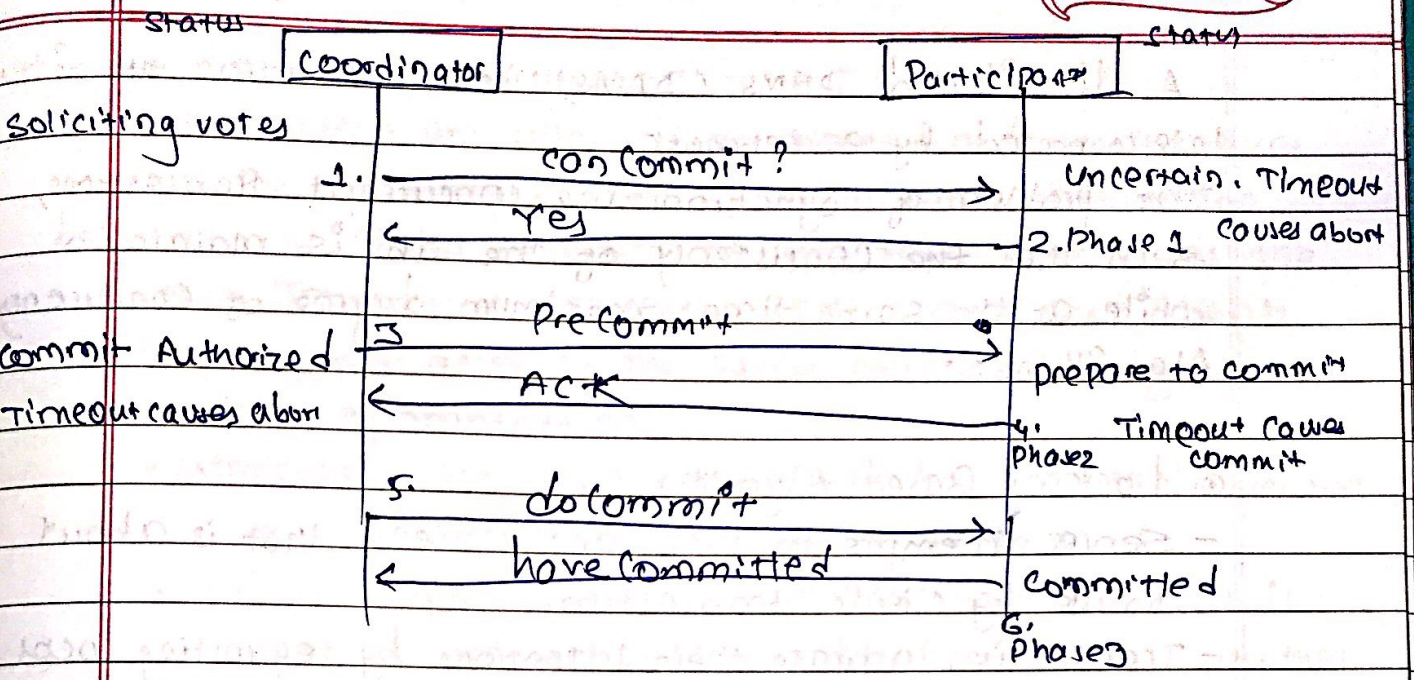
Phase 3: If after a participant member receive a preCommit message, Coordinator fails or times out, the participant member goes forward with the commit



(a): Coordinator



b: Participant.



6[a] What is Nested Transaction? Explain the Concurrency Control mechanism for Distributed transaction

- A nested transaction is a trasaction that is created inside another transaction. A nested transaction's purpose is to support transactions in stored procedures that can be called from a process already in a transaction or from a process that has no active transaction.

Example:

```
BEGIN TRAN Tran1
GO
BEGIN TRAN Nested Tran
GO
INSERT INTO Table1 Default values
GO 100
COMIT TRAN Nested Tran
SELECT * FROM Table1

-ROLLBACK TRAN Tran1
```

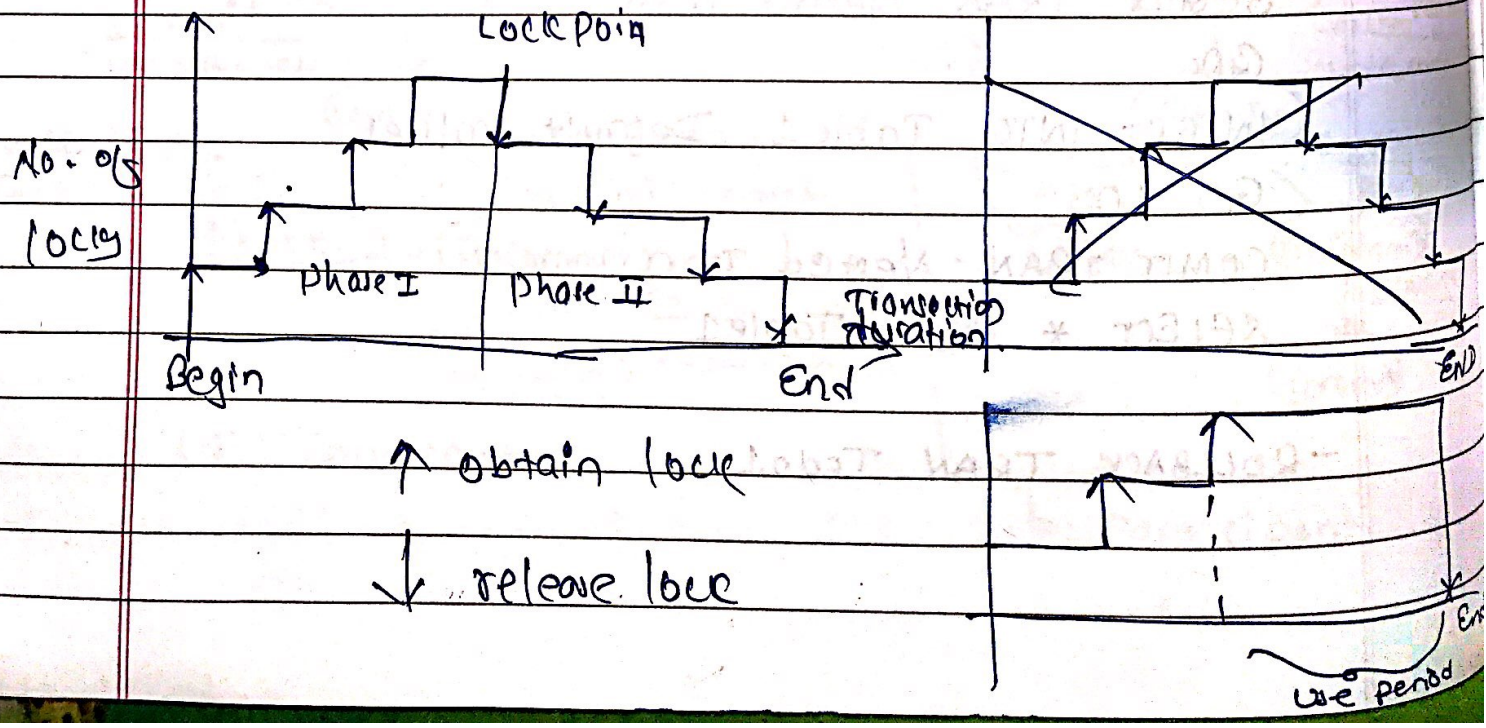
A distributed DBMS (DDMS) is a collection of sites interconnected by a network.
The problem of synchronizing concurrent transactions such that the consistency of the db is maintained while, at the same time, maximum degree of concurrency

Algorithms:

a. Locking Based Algorithm:
- Server attempts to lock any object that is about to use by client's transaction.
- Transaction indicate their intentions by requesting locks from the scheduler (called lock manager).
- locks are either read lock (shared lock) or write lock (exclusive lock)

2PL (Two phase-locking):
- A transaction locks an object before using it.
- when an object is locked by another transaction, the requesting transaction must wait.
- when a transaction release a lock, it may not request another lock

Strict 2PL: hold locks until the end



b. Optimistic Concurrency control:
- Transaction are allowed to proceed as though there were no possibility of conflict with other transaction until the client completes its task and issues the close Transaction request. Likelihood of two clients transactions accessing the same object is low.

3 Phase of transaction:
- Working phase: Each transaction has a tentative version of each of the objects that it updates
- validation phase: when the closeTransaction request is received, transaction is validated to establish whether or not its Operation on objects conflict with other transaction.
- Update phase: change in tentative version are made permanent if transaction is validated

advantage: deadlock free, maximum parallelism
disadvantage: Rerun transaction if aborts, not used widely

C. Timestamp-based concurrency control:
- Each transaction T_i^0 is given timestamp $ts(T_i^0)$
- If T_i^0 wants to do an operation that conflicts with T_j
  - Abort T_i if $ts(T_i^0) < ts(T_j^0)$
- when a transaction aborts, it must restart with new timestamp (current timestamp).
- Two values for each data item X
  - max-rts(x): max timestamp of a transaction that read X
  - max-wts(x): max timestamp of a transaction that write X
- $Read_i^0(x)$:
  If $ts(T_i^0) < max\text{-}wts(x)$ then abort T_i^0
  else
    Perform $R_i^0(x)$
    $max\text{-}rts(x) = max(max\text{-}rts(x), ts(T_i^0))$

- write_i(x)
  - If $ts(T_i) < max\text{-}rts$ or $ts(T_i) < max\text{-}wts(x)$
    abort T_i
  - else perform $write_i(x)$
    $max\text{-}wts(x) = ts(T_i)$

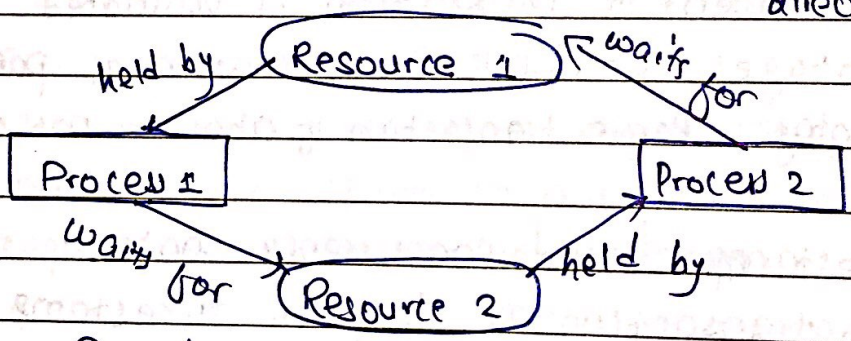
6b] What is distributed deadlocks? Explain obermarck's Path pushing algorithm for Distributed deadlock detection.

Distributed Deadlock:

- Permanent blocking of a set of processes that either compete for system resources or communicate with eachother
- No node has complete and up to date knowledge of the entire Distributed system

Wait For Graph (WFG):

Nodes: process in system
direct edge: wait for blocking relation



cycle means a deadlock

Fig: illustrating a deadlock

Starvation: A process' execution is permanently halted.

- Resource deadlock: set of deadlocked processes, where each process where each process wait for a resource held by another process (eg: I/O resource on a server).
- Communication deadlock: set of deadlocked process, where each process waits to receive msg from other processes in the set.

Obermarck's Path-Pushing algorithm:
- Individual sites maintain local wait for graph (WFG)
- A virtual node 'x' exists at each site represents the external processes.

Detection process
- Case 1: If site Sn finds a cycle not involving 'x':
  — Deadlock exists;
- Case 2 If site Sn finds a cycle involving 'x'
  — deadlock possible;
- If Case 2 happens:
  * Site Sn sends a message containing its detected cycles to other sites. All sites receive the message, update their WFG and re-evaluate the graph.
  • Consider site Sj receives the message.
    ✓ Site Sj checks for local cycles; if cycle found not involving x (of Sj)
      — deadlock exists
    ✓ If Site Sj finds cycle involving x, it forword the message to other sites
  • Process continues till deadlock found. or all sites checking is done.

2016 Spring

1[a] List key characteristics in DS. Describe Transparency and openness in detail:

- The key characteristics of DS are:
  1. Resource sharing
  2. Openness
  3. Concurrency
  4. Scalability
  5. Fault Tolerence
  6. Transparency

- As we have seen DS is a collection of autonomous systems, which are connected together by means of network.

a. Transparency:

- Transparency refers to the characteristics of communication system being perceived by users as a single entity rather than as a collection of autonomous systems in cooperations, with the users being unaware of the processes occuring underneath.
- Transparency makes the operation in the eyes of the user to be more friendly, easy or simply transparent. The user should be unaware of where the services are located and also the transferring from a local machine to a remote one should also be transparent.
- As DS are complex, steps have to be taken so that the complexity doesnot hinder the user from or make the user worry about using the system.
- The following are the different transparencies in DS:

i) Access Transparency: clients and users should be unaware of the distribution of files interms of specific servers or physical location; the files simply have to be available and accessible whenever they are required. Eg: file system in NFS, ~~Accessing~~ Navigation of web.

ii) Location Transparency: Hides the location of resource, the pathnames should remains the same. Name should contain no information about the object's physical location.

iii) Performance Transparency: The system can be reconfigured to improve performance as the load fluctuates but this process should be transparent to the user who is using the system.

iv) Migration Transparency: Information and processes can be migrated or moved from one physical server to the next within the system, with the user being unaware that this is going on. This is related to performance transparency as this is often done for load balancing to improve performance.

v) Relocation Transparency: Hide that a resource may be moved to another location while in use.

vi) Replication Transparency: Hide that a resource is replicated.

vii) Concurrency Transparency: Hide that a resource may be shared by several competitive users.

viii) Failure Transparency: Hide the failure and recovery of a resource.

b. Openness:

- Be able to interact with services from other open systems, irrespective of the underlying environment. It makes independent from heterogeneity.
  - system extension, re-implementation via standards
  - portability, interoperability, conformance to well defined interfaces.
- A system is open if it is easy to extend or integrate

- system should conform to well-defined interface
- Eg: Internet, Git are open distributed system.
- Services in DS are generally specified through interfaces, which are often described in an IDL.
- New components have to be integrated with existing components.

1(b) Define IPC. Explain request reply Communication with the different procedures involved.

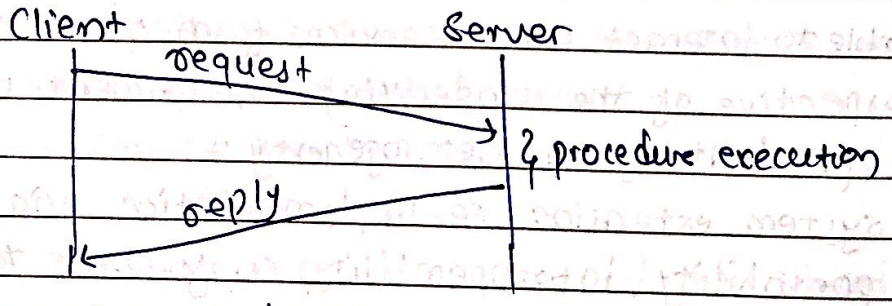
Interprocess Communication (IPC) is a mechanism that allows the exchange of data between processes. IPC is a set of programming interfaces that allow a programmer to coordinate activities among different program processes that can run concurrently in an operating system.

IPC methods: pipe, socket, RPC, Message passing, Shared Memory

but Shared Memory and Message passing are two methods

- Request reply is one of the basic methods computers use to communicate with each other, in which client's computer sends a request for some data and server responds to the request. Eg: browsing a webpage, telephone call.

The different procedures of request reply Communication are:

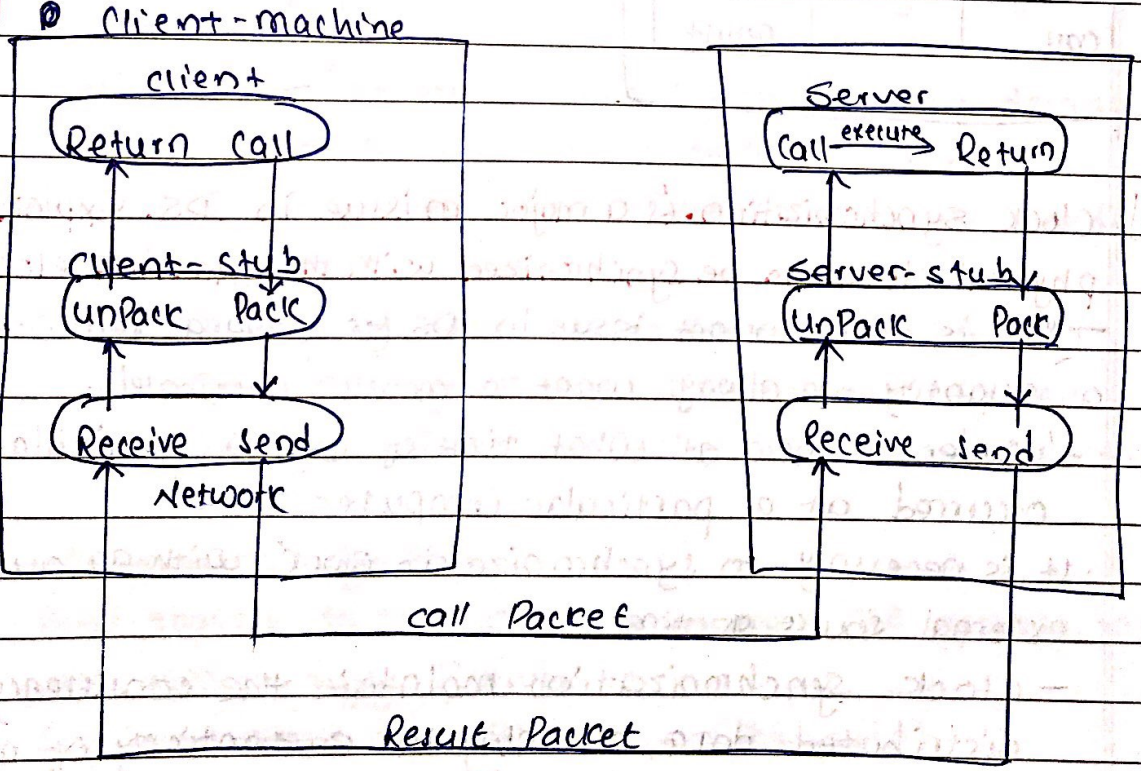


- It is useful for designing systems which involve simple RPCs. In a simple RPC all the ~~request~~ arguments and result fit in a single packet buffer while call duration and interval between calls are short.

Example of RR are: RPC, RMI, Mobile code, etc.

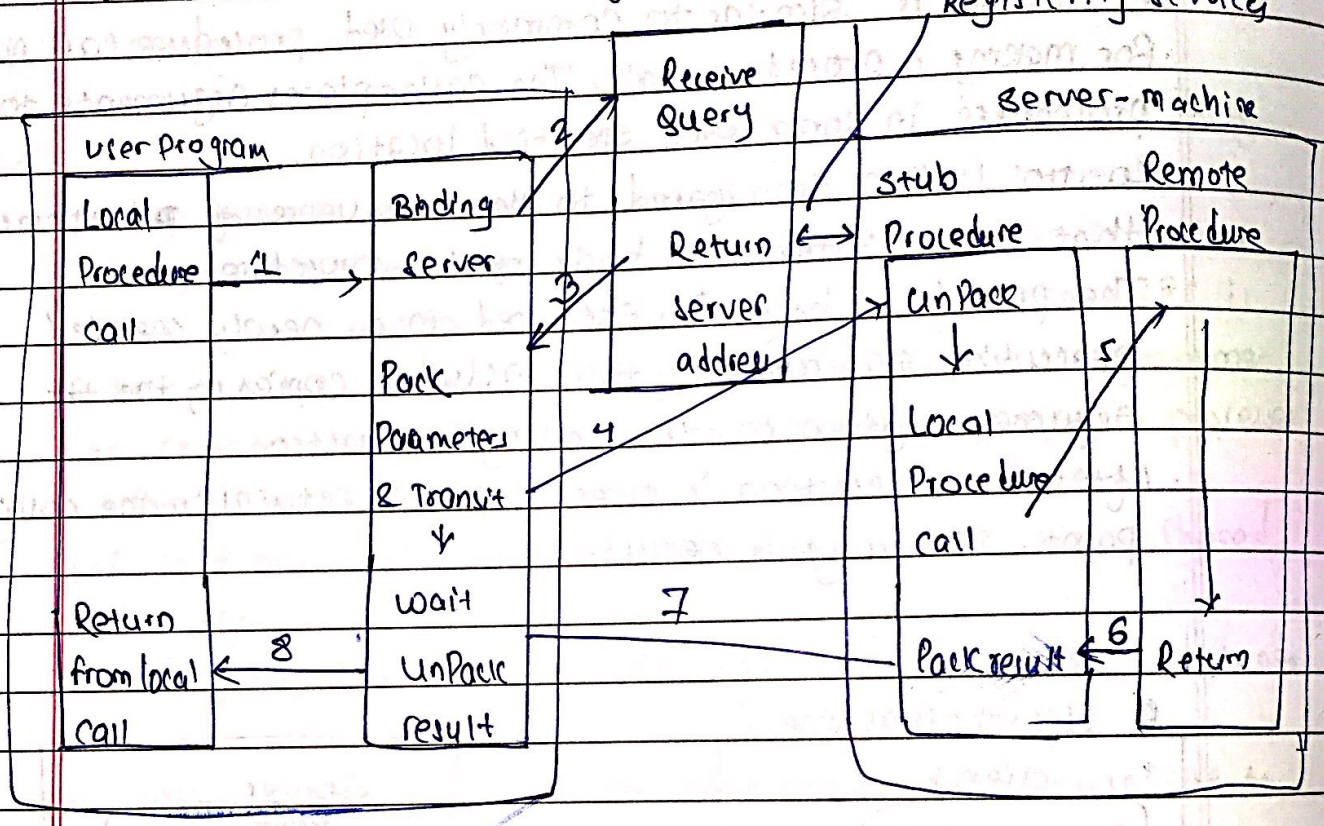
21Q] Interface process communication handling and Binding are main task of RPC implementation. Explain these tasks with neat sketch.

→ RPC model is similar to commonly used procedure call model.
- For making a procedure call, the caller places arguments to the procedure in some well specified location.
- control is then transfered to the sequence of instructions that constitutes the body of the procedure.
- The procedure body is executed in a newly created execution environment that includes copies of the ~~the~~ arguments given in the calling instruction.
- After the execution is over, control returns to the calling point, returning a result.



Binding is explain in Note:

The process of finding out or assigning the port and corresponding system (client and server) is called binding. It determines remote procedure and machine on which it will be executed.
- checks the compatability of the parameter passed



2[b] Clock synchronization is a major issue in DS. Explain how a physical clock can be synchronized with the standard clock.

→ Time is an important issue in DS for several reason. Time is a quantity we always want to measure accurately.
- Inorder to know at what time of day a particular event occurred at a particular computer
- It is necessary to synchronize it clock with an authorative external source of time.
- clock synchronization maintain the consistency of distributed data, checking the authenticity of a request sent to a server and eliminating the process of duplicate updates.

- In centralized system, there is only ~~one~~ single clock, process gets the time by simply issuing a system call to the Kernel.
  - when one process after that, tries to get the time, it will get the higher time value. In such systems, there is a clear ordering events, no ambiguity about times.
- In Ds, there is no global clock or common memory. Each processor has its own internal clock (own notion of time). These clocks can easily drift apart by several seconds per day, accumulating significant errors over time. Because Different clocks ticks at different rates, they may not remain always synchronized although they might be synchronized when they start.
  - Accurate time for Transactional integrity, system and Network, wide logging, auditing, forensics play critical role.
  - Events must be correlated wherever they are generated.

→ Physical clocks are those clocks which must not deviate from the real time by more than a certain amount.
we can synchronized physical clock with the standard clock by :

1. Cristian's algorithm - centralized system
2. Berkeley algorithm - centralized system
3. Network Time protocol - Distributed system.

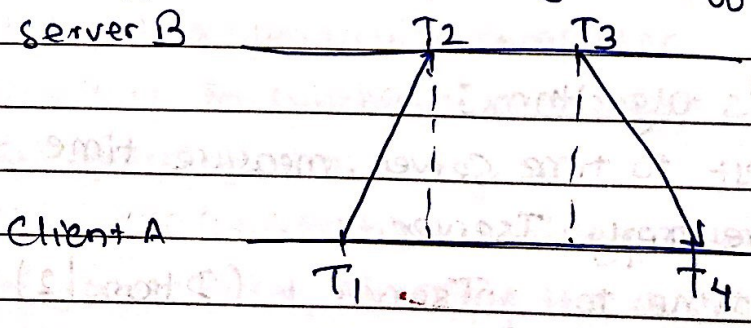
a. Cristion's algorithm :

- Send request to time server, measure time Dtrans taken to receive reply Tserver
- Set local time to Tserver + (Dtrans/2)
  - improvement, take average Tserver of several request.
- Assumptions: - Network delay is fairly consistent.
  - Request-reply take equal amt of time.

- to offset variations in time delay - client may average over several request.
- Problems:
  - doesnot work if time server fails
  - Not secure against malfunctioning time servers.

b. Berkeley algorithm
- choose a coordinator computer to act as the master.
- Master periodically polls the slaves - the other computers whose clocks should be synchronized to the master
  - Slaves send their clock value to master
- Master observes the transmission delays, and estimate their local times.
  - master averages everyone time (including its own)
  - Master takes a fault tolerent average i.e - it ignore readings from clocks that have drifted badly or outside the range of other clock.
  - Master sends to each slave the amount (positive or negative time) by which it should adjust its clock.

c. Network Time Service Protocol (NTP)
- UDP (Connectionless), Multi-cast: for LANs (Low accuracy)
- NTP client ask all NTP servers what time it is.
- The different server will give different answers,



A knows $T_4 - T_1$ from its own clock
B reports T_3 and T_2 in response to NTP request

- A computes total transit time of : $(T_4 - T_1) - (T_3 - T_2)$
- One way transit time is: $\frac{(T_4 - T_1) - (T_3 - T_2)}{2}$

- B's clock read T_4 as:

$$T_3 + \text{one way transit} = T_3 + \frac{(T_4 - T_1) - (T_3 - T_2)}{2}$$

$$= \frac{2T_3 + T_4 - T_1 - T_3 + T_2}{2} = \frac{(T_4 - T_1) + (T_2 + T_3)}{2}$$

The difference between B and A clocks at T_4 is

$$\frac{T_4 - T_1 + T_2 + T_3 - T_4}{2} = \frac{T_4 - T_1 + T_2 + T_3 - 2T_4}{2}$$

$$= \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$$

3[9] Why is name server used? Explain the working principle of a typical name server.

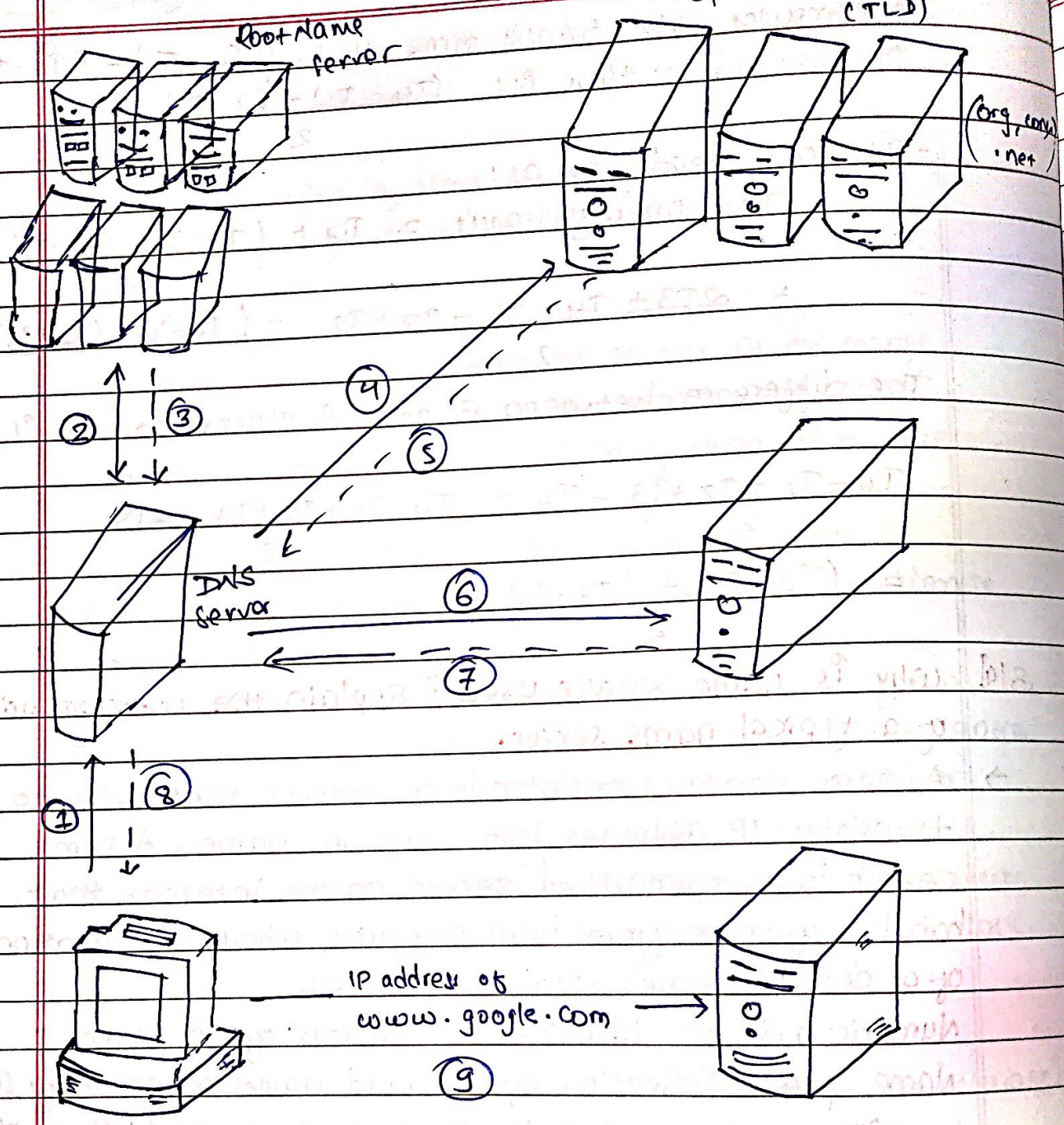
→ A name server ~~translater~~ is a server that helps to translates IP addresses into domain names. A name server is a specialized server on the internet that handles querries from local computer, about the location of a domain name's various services.

Numeric addresses identify the location of an object.

Name space : Collection of all valid names recognized by servic with a syntax; rules for resolving names (left to right)

Binding: Association between an object and a name

- Name servers stores the collection of naming contexts
- Major operation of a name server : Name resolution, that is look up attributes from a given name.
- operations: creating new bindings, Name →  server
  deleting bindings and Attribute →
  listing bound names, adding and deleting contexts.



1: Issues a DNS query to ask for the IP address of www.google.com
2. Issues a query to a name server
3: It returns the IP address of the TLD.
4: Issue a querry and sent to the TLD servers.
5. Reply with ns1.google.com and its IP address
6:- Issues an another query and sent to the ns1.google.com
7: return the IP address of www.google.com.
8: hit the IP address of google.com.