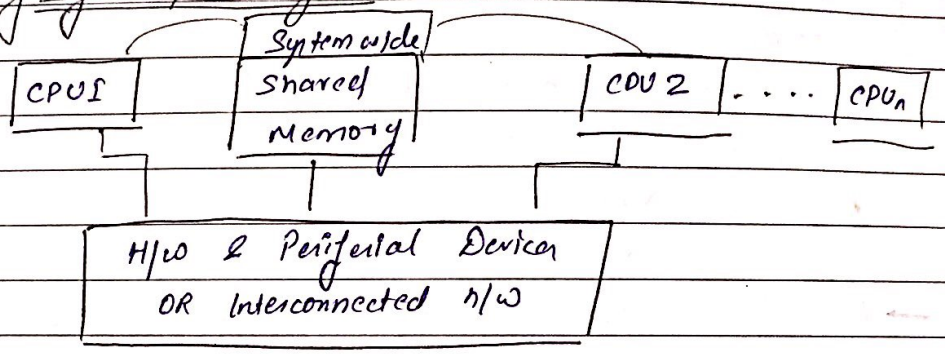
Sunil
Page
Date 11/08/18

* COMPUTER ARCHITECTURE :-

↳ Tightly Coupled System
↳ Loosely coupled system.

These architecture consist of interconnected multiple processors.

① Tightly Coupled System :-



• It has single address space

② Loosely Coupled System :-

• Every system is having its own memory space that is its local memory.

DEFINITION OF DISTRIBUTED SYSTEM :-

A distributed computing system is collection of independent computer that appears to its user as a single coherent

system.

OR A collection of processors interconnected by a communication network in which, -

- Each processor has its own local memory and other peripheral devices.
- Communication b/w any two processors takes place by message passing.
- For any particular processor its own resources are local resources and resources on the other processor will appear as remote resources.
- Together a processor and its resources are usually referred as a node or a site or a machine of the distributed computing system.

→ In distributed computing system is organised as middle ware.

→ Middle ware layer extend ~~up~~ over multiple machines.

18/08/18

Challenges In Distributed System! -

1. Heterogenity :-
2. Openness
3. Security - Single system is more secure than distributed system.
4. Scalability
   Scale up & scale down can be done but it cause problem in policies.
5. Failure Handling
6. concurrency
7. Transparency → It should be transparent for every single node in the system.

Sunil
Page .....
Date .....

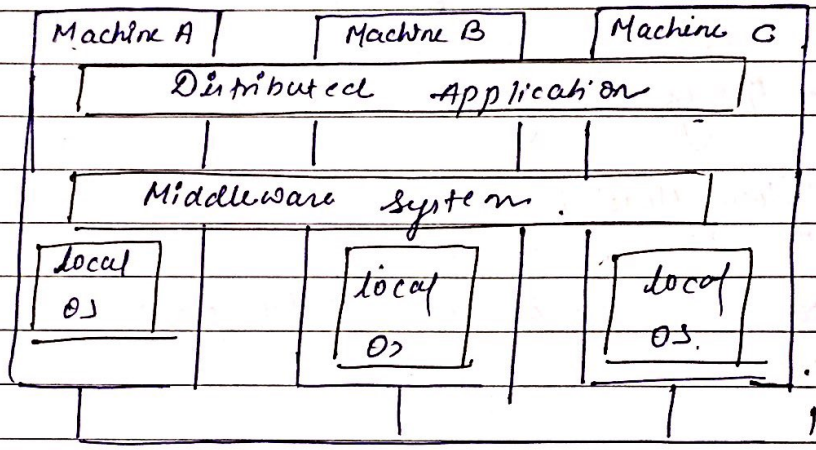
Different models in distributed system :-

① Architectural Model :-

- An architectural model in distributed system is concern with replacement of the parts & the relationship b/w them.

→ Software layer :-

Middleware layer is responsible for performing all the task of distributed system.



20/08/18

② → System Architecture :-

It includes division of responsibilities between system components

→ Types of System Architecture :-

(i) Client-Server
(ii) Peer to Peer
(iii) Interface an object.

→ Design Requirements :-

① Performance ② Quality of Service ③ Use of Replication and caching

by message passing.

Sunil
Page .....
Date .....

→ Dependibility Issues :-

(2) Fundamental / Fonctional Model :-
These are concern with a more formal discription of the properties that are common in all the architectural models

→ Interaction Model :-
In the interaction model we deal with the performance and also deal with difficulty of setting time limitations in a distributed system.
Eg - Message delivery.

→ Failure Model :-
It attempts to gives a precise information of the faults that can occur / that can be executed by processes & common channels.

→ Security Model :-
It discuses the possible ~~threats~~ threats to processes and to the communication channel.
Eg - Protecting objects, Securing processes & their interactions.

23/08/18

Limitation Of Distributed System :-
- Problem of local clk and global clk.
Then logical clk here we are giving the number to the events.

- Shared Memory

Sunil
Page .....
Date .....

→ INHERIENT LIMITATION OF DISTRIBUTED SYSTEM :-

① Absence of global clock —

In a distributed system concept of global time does not exist.

It is a limitation for distributed system. We cannot solve this problem by either having a clock common to all the computer in the system or having a senquarnised clock in each computer system because if we use a global clock (common clock) then two diff processes can observed a global clock value at diff instance due to unpredictable message transfer delays.

On the other hand if we provide each computer in a system a physical clock and try to senquarnise them then physical clock can drift from the physical time. and the drift rate may vary from clock to clock due to technical limitation

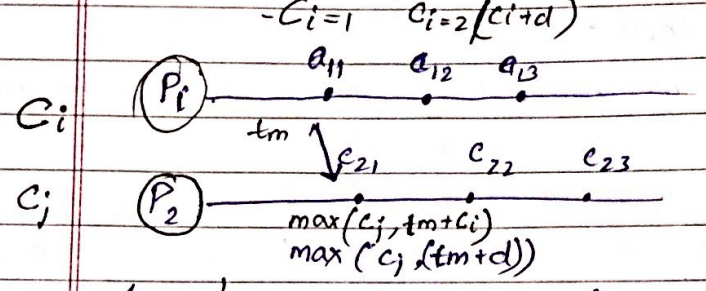
Due to absence of global time it is difficult to reason about or comment about the temporal order of the events in distributed system.

② Absence of Shared Memory :-

In distributed system do not share common memory so an upto date state of the entire system is not available for any individual process

→ Lamport's Logical clock :-

We donot care about the actual time for the occuring of an event.

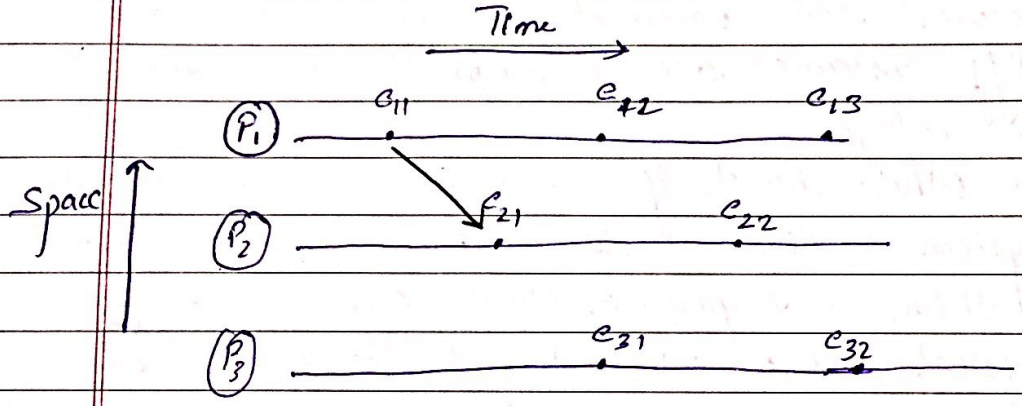


where d is a difference

'→' : happen before relation

If their is nor happen before relation then it is concurrent events

24/08/18



Lamport proposed a scheme to order events in a distributed system using logical clock. Due to the absence of perfectly synchchised clock and global time in system, the order in which 2 event occur at 2 different computers can't be determined based on the local time at which they occur.

Happen before relation (' → '):- It captures the casual dependencies b/w events that is whether 2 events are casually related or not. The relation (→) is defined as follows —

(1) $a \rightarrow b$: a happen before b

If a and b are events in same process & a occured

Sunil
Page .....
Date .....

before b
eg – $e_{11} \rightarrow e_{12}$

(2) $a \rightarrow b$:- If a is an event of sending a message 'm' in a process & b is an event of receipt of same msg 'm' by another process.
eg – $e_{11} \rightarrow e_{21}$

(3) If $a \rightarrow b$ & $b \rightarrow c$ then $a \rightarrow c$ is referred as casual affects that is '$\rightarrow$' relation is transitive.

All the events that can be order by ('$\rightarrow$') are refered as casual affects.

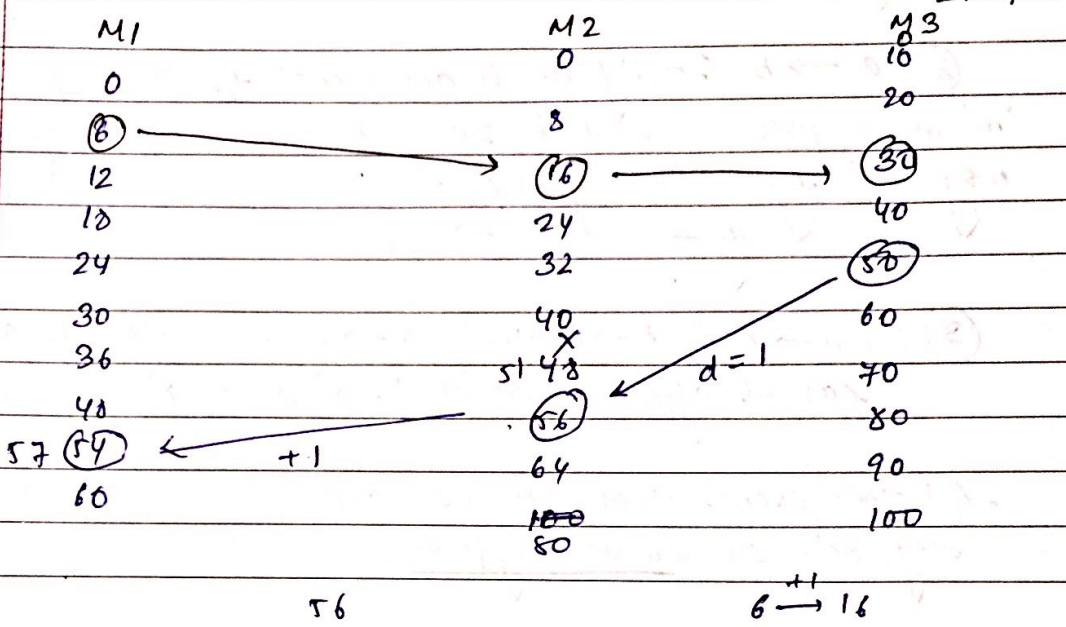
Concurrent Events :- Two distinct events 'a & b' are said to be concurrent event denoted by $a \| b$ if $a \rightarrow b$ & $b \rightarrow a$ i.e concurrent event do not casually affect each other.

Whenever $a \rightarrow b$ holds for 2 events a & b their exist a path from a to b which moves only forward along with a time axis in the space time diagram.

- In a space-time diagram e_{11}, e_{12} & e_{13} are events in the process P_1 & e_{21}, e_{22} are events in the process P_2. Arrow represents message transfer b/w the process. For eg – $e_{11} \rightarrow e_{21}$ corresponds to a message m send from process P_1 to P_2. e_{11} is the event of sending the message m at P_1 & e_{21} is the event of receiving the same

message at P_2.

28/07/18



56

$\therefore 56+1$

$\max(6+d, 16) = 16$.

30/08/18

VECTOR LOGICAL CLOCKS :-

- In order to realise the relation '$\rightarrow$' lamport introduce the following system of logical clocks
- There is a clock C_i at each process P_i in the system C_i can be though as a function that assigns a number $C_i(a)$ to any event a at P_i. The number assigned by system of clocks have no relation with to physical time & hence the name logical clock.
- The logical clock takes monotonically increasing values There clock can be implemented by counters.
- The happen before relation '$\rightarrow$' can now be realised by using the logical clock if the following two conditions are satisfied.
  1. For any 2 events in $(a \& b)$ in process P_i

Sunil
Page .....
Date : .....

if 'a' occurs before 'b' then $C_i(a) < C_i(b)$

(2) If a is an event of sending a message 'm' in a process P_i and 'b' is the event of receiving the same ~~prior~~ message 'm' at process P_j

$$C_i(a) < C_j(b)$$

The following implementation rules for the clocks guarantees that ~~the~~ the clock ~~the~~ satisfy the correctness condition (1) & (2).

(1) Clock C_i is incremented b/w any 2 successive events in process P_i

$$\boxed{C_i = C_i + d} \quad (d > 0)$$

If 'a' and 'b' are 2 successive events in process P_i ~~then~~ & 'a → b' then $\boxed{C_i(b) = C_i(a) + d}$

(2) If event 'a' is sending the message 'm' by process P_i then 'm' is assigned a timestamp t_m. On receiving the same message 'm' by Process P_j, C_j is set to a value greater than or equal to its present value and greater than t_m

~~C_j = max~~ $\boxed{C_j = \max(C_j, t_m + d)}$

→ We are solving the problem of casually ordered events

VECTOR CLOCKS :-

Lamport timestamp lead to a situation where all events in a distributed system are totally ordered with the property that if event 'a' happened before event 'b' then 'a' will also be positioned in that ordering before 'b' i.e $\boxed{C_i(a) < C_i(b)}$ However with lamport timestamp nothing can be said about the relationship b/w 2 events 'a' & 'b' by merely comparing their time values C_a and C_b respectively in other words if $C_a < C_b$ then this doesn't necessarily imply that 'a' ~~in~~ deed happen before 'b' some more information is required for that.

when we are prior info^n. using that info^n and updating that info^n. and using that. is known by piggy backing.

Page .....
Date 03/09/18

The problem with Lamport Timestamp is it donot capture causality. This issue can be captured by means of vector timestamp

- A vector timestamp 'VT(a)' assign to an event 'a' has the property that if VT(a) < VT(b) for some event 'b' then event 'a' is known to causally precede event 'b'

- Vector timestamp are constructed by letting each proces 'Pi' maintain a vector 'Vi' with the following 2 properties ——
  ① Vi[i] is the number of events that have occured so far at 'Pi'.
  ② If Vi[J] = K then, 'Pi' knows that 'K' event has occured at 'Pj'

→ The 1st property is maintain by incrementing Vi[i] at the occurance of each new event that happens at proces 'Pi'
→ The 2nd property is maintained by piggy backing vectors along with messages that are sent.
→ When 'Pi' sends message 'm' it also sends along its current vector as a timestamp 'Vt'
→ In this way a receiver is informed about that no. of events that have occured at 'Pi'. However is that is that receiver is told that how many events at other proces have taken place before 'Pi' sends message 'm'. In other words timestamp 'Vt' of 'm' tells the receiver how many events in other proceses have preceded 'm', and on which event 'm' may casually depend.

Advanced OS by Bhargavi Potluri

Sunil
Page .....
Date .....

→ When process 'P_j' receives message 'm' it will adjust its own vector by setting each entry $V_j[k] = \max(V_j[k], vt[k])$
The vector now reflects the no. of messages that 'P_j' must receive to have atleast seen the same messages that preceded the sending of 'm'. Here after '$V_j[j]$' is incremented by '1' representing the event of receiving the next message. With the slight adjustment vector timestamp can be used to currently casual message delivery. In particular a message 'r' is delivered only if

(1) $Vt(r)[j] = V_k[j] + 1$
(2) $Vt(r)[i] \leq V_k[i]$ for all $i \neq j$

⇒ The 1st condition states that 'r' is the next message that 'P_k' was expecting from process 'P_j'
⇒ The 2nd condition states that 'P_k' has not seen any message that were also not seen by 'P_j' when it send message 'r'. In particular this means that 'P_k' has already seen message 'a'

→ GLOBAL STATE :- 7/9/18.

Uses of Global state
(1) Deadlock Detection
(2) Termination Detection

In the global state we take snapshot of the entire distributed system and share the snapshot with each node of distributed system so that nodes will have information of all the resources and state of the processes running in a distributed system.

With the help of global state we can predict the behaviour of entire distributed system that all the processes in the

distributed system are terminated successfully or the entire systems have entered into deadlock state.

→ Casual Ordering of Messages

Election Algorithm —

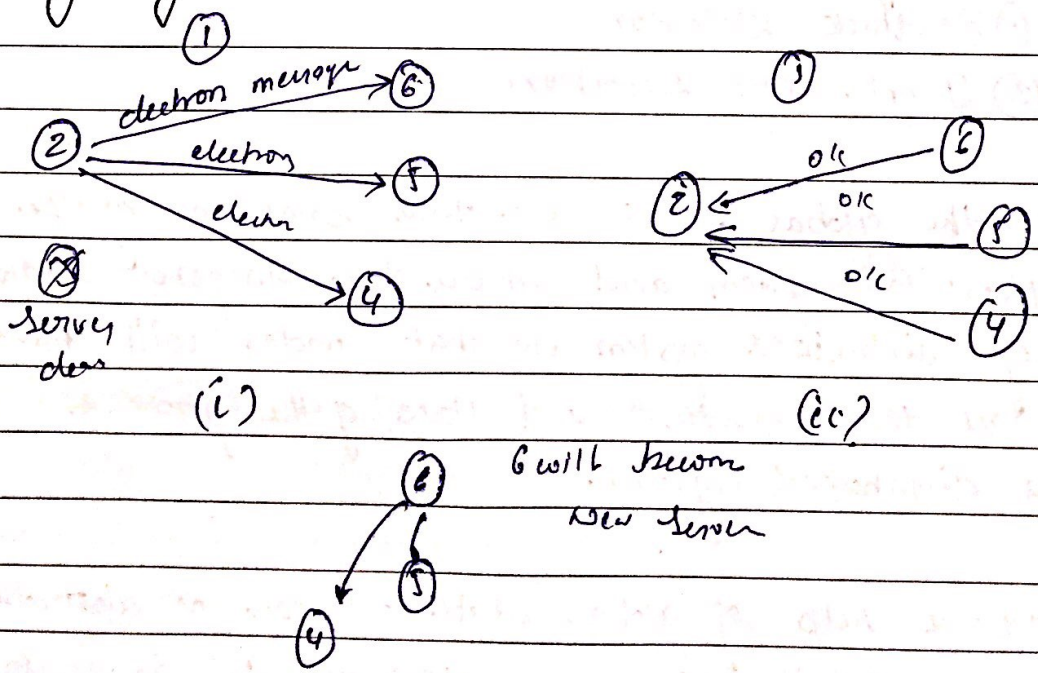
(1) The Bully Algorithm —

When any process notices that the co-ordination is no longer responses to a message / request it initiates an election. The process p holds an election as follows —

→ P sends an election message to all the processes with higher number if no one responses with 'OK' message P uses the election and becomes co-ordinator

→ If one of the here up answer it takes over and there jobs are done

(2) Ring Algorithm —



Sunil
Page .....
Date .....

→ DISTRIBUTED MUTUAL EXCLUSION

Requirements of Mutual Exclusive Algorithm –
The primary objective of a mutual exclusive algorithm is to maintain mutual exclusive that is to guarentee that only one request ques the critical section (C.S) at a time.

In addition, following characteristics are considered importan. in an mutual exclusive algorithm.
(1) Freedom from deadlock
(2) Freedom from ~~satruation~~ starvation
(3) Fairness
(4) Fault Tolerance

(1) Freedom from deadlock – Two or more sides should not endlessly wait for messages that will not arrive

(2) Freedom from starvation – A site should not be forced to wait for infinite time to execute critical section while other sites are repetedly executing critical section

(3) Fairness – Fairness dictates that request must be executed in the order they were made since, The physical global clock does not exact time is determined by logical clock Note that fairness implies freedom from starvation but not vice-versa.

(4) Fault Tolerence – A mutual exclusive algorithm is fault tolerent if in the wake of a failure it can recognize itself so that it continues to function without any distraction.

→ How to Measure the Performance ?

① No of messages
② Synchronization Delay.
③ Response Time
④ System Throughput -

→ TOKEN BASED & NON-TOKEN BASED ALGORITHM

① Non - Token Based Algorithm : -

- In Non-token based mutual exclusive algorithms a site communicates with a set of other sites to arbitrate who should execute the critical section next.
- For a site Si, request set Ri contains ID's of all the site from which site Si must acquire permission before entering into critical section.
- Non - token based mutual exclusion algorithms use timestamps to order request for the critical section, and to resolve the conflict b/w simultaneous request for the critical section.
- In these algorithms logical clocks are maintained and update according to lamport's scheme. Each request for the critical section gets a timestamp & smaller timestamp request have priority over larger timestamp request.

Eg - Lamport, Richart - Agrawala, Maekawa Algorithms

Sunil
Page .....
Date .....

(2) TOKEN BASED ALGORITHMS :-

- In token based algorithms the unique token is shared among all the sites. A site is allowed to entered into critical section if its possess the token.
- Token based algorithm use 'sequence number' instead of timestemp. Every request for the token contains a sequence number and sequence number of the site advance independently.
- A site increments it sequence number counter every time it makes a request for the token

Eg - Suzuki - Kasami's
Singhal's heuristic
Raymond's tree
Broadcast Algorithm

Sunil
Page
Date 14/09/18

NON-TOKEN BASED –

(1) Lamport's Algorithm :-

Lamport was first one to give student mutual exclusive algorithm as an illustration of its clock synchronization scheme. in lamport algorithm

~~In lamport algorithm~~

$\forall i : 1 \leq i \leq N :: R_i = \{S_i, S_2 \cdots S_N\}$

Every site - S_i keeps a queue : request queue (rq_i) which contains mutual exclusive request ordered by their timestamp.

Algorithm –

(a) Requesting the critical section

(i) When a site S_i wants to enter in critical section it send a request message REQUEST (ts_i, i) to all the sites in its request sent (R_i) and places the request on request_queue where (ts_i, i) is timestamp of the request.

(ii) When a site S_j receives a request REQUEST(ts_i, i) message from the site S_i, it returns a timestamp reply to S_i and places site S_i request on request queue.

(b) Executing the critical section: when the following 2 conditions. hold.

(i) S_i have received a message with timestamp larger than (ts_i, i) all the other site

(ii) S_i's request is at the topic of request queue.

(c) Releasing the critical section.

(i) Site S_i upon releasing the critical section removes its request from the topic of its request queue and sends a timestamp release message 'RELEASE' to all the sites in its request set

receives the

(iii) When a site S_j release message 'RELEASE' from S_i, it removes S_i request from its request queue.

→ When a site removes a request from its request queue, its own request may come at the top of the queue - enabling it to enter into critical section. The algorithm executes critical section request in the increasing order of timestamp

(2) TOKEN BASED :-

(1) SUZUKI – KASAMI'S BROADCAST ALGORITHM :-

Algorithm :-

(a) Requesting the critical section

(i) If the requesting site S_i does not have token then it increments its sequence no. 'RN$_i$[i]' and sends the request REQUEST (i, S_n) message to all the site (S_n is the updated value of "RN[i]")

(ii) When a site S_j receive this message it sets RN$_j$[i] to max (RN$_j$[i], S_n). If S_j has the idle token then it sends the token to S_i, if RN$_j$[i] = LN[i] + 1

(b) Executing the critical section

(iii) Site S_i executes the critical section when it has received the token.

(c) Releasing the critical section

Having finished the execution of critical section site S_i take the following options —

(iv) It sets LN[i] element of the token array equal to RN[i]

If one process stop then it is starvation
If whole system works stops then it is deadlock

Sunil
Page .....
Date .....

(v) For every site S_j whose ID is not in the token queue, it appends its ID to the token queue if $RN_i[j] = LN[j] + 1$

(vi) If token queue is non-empty after the above update then it deletes the ~~top~~ top site ID from the queue and sends the token to the site indicated by the ID.

→ **Deadlock Detection in Distributed System :-**

Deadlock can be dealt with using any of the following 3 strategies.

① **Deadlock Prevention :-** It is commonly achieved by either having a process acquire all the needed resources simultaneously before it begins the execution. or by preempting a process that holds the needed resource

② **Deadlock Avoidance :-** This approach in distributed system a resource is granted to a process if the resulting global system is safed.

③ **Deadlock Detection :-** It requires an examination of the status of the process resource interaction for the presence of a deadlock condition. To resolve the deadlock we have to abort the ~~prior~~ deadlock process.

Sunil
Page .....
Date 17/09/18

Centralized Deadlock Detection Algorithms :-

A central system has the all information about all the resources

Distributed Deadlock Detection Algorithms :-

- In this algorithms, all sites collectively cooperates, to detect a cycle in the state graph that is likely to be distributed over several sites of the system.
- DDDA can be divided into 4 classes.—
  (1) Path Pushing
  (2) Edge Chasing
  (3) Diffusion computation
  (4) Global state Detection.

Path Pushing –

In path pushing algorithms the wait-for dependency infn of the global WFG (wait for graph) is ~~designated in the form~~ disseminated in form of path.

Edge Chasing :-

In edge chasing algorithm special messages called as 'probes' are circulated among the edges of ~~wait~~ WFG to detect a cycle. When a blocked process receives a probes its propagates the probe message along its outgoing edges in the WFG.

A process declares a deadlock when it receives a probes initiated by it only.

Sunil
Page .....
Date .....

Diffusion computation —

- These algorithm makes use of eco-algorithms to detect deadlock.
- To detect deadlock a process sends out query messages along all the outgoing message wfg. Then query queries are recursively propagated through the edges of graph.
- Queries are discarded by a running process and echoed by a blocked process. In the following way —

→ When a block process receives first query message for a particular deadlock detection initiated it doesnot send a reply until it has received by a reply message from every site send by this process for all subsequent query for this deadlock detection it immediately sends back a reply message. The initiater of the deadlock detection detects a deadlock when it receives a reply from every query it sents.

Global State Detection —

This based deadlock detection algo explict the following facts —

→ A consistent snapshot of a system can be contained without freezing the underlined computation.

→ A consistent snapshot may not represent the system state at any moment of time, but if a stable property holds in the system before the snapshot collection is initiated, this property will still hold in the snapshot.

Therefore distributed deadlocks can be detected by taking the snapshot of a system and examining it for the condition of a deadlock.