Date: / /

→ Challenges in Distributed Systems

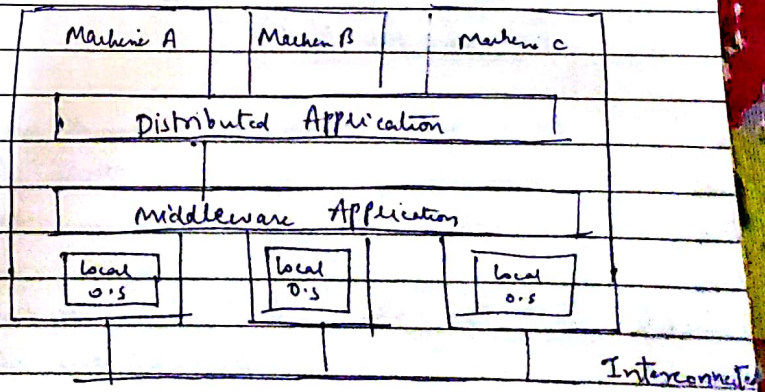
1) Heterogeneity
2) openness
3) Security
4) Scalability
5) Failure Handling
6) Concurrency
7) Transparency.

→ Distributing Models

I) Architectural Model :-

An Architectural model in distributed system is concerned with placement of its parts and the relationship b/w them

(i).1→ software layers.



Date: / /

(ii) System ~~Model~~ Architecture

It includes division of responsibilities b/w system components

Types of System Architecture

(i) client - Server
(ii) Peer to Peer
(iii) Interfaces & Object

→ Design Requirements

1) Performance
ii) Quality of service
(iii) use of replication & caching

→ Dependability Issues

II) Fundamental Model

Fundamental models are concerned with a formal description of the properties that are common in all the architecture models

(i) Interaction model:- In this model we deal with the performance and also deal

Date: / /

with the difficulty of setting time limitations in a distributed system. for eg - Message Delivery

(ii) Failure Model :-

This model gives attempts to give the precise specification of the faults that can occur / or can be emitted by processes & communication channels.

(iii) Security Model :

It discusses the possible threats ~~made~~ to be the processed and to the communication channels eg:- protecting objects, securing process and their interactions.

→ lampord logical clock

Date: / /

Date: / /

Logical clocks :-

In order realize the relation ~~so~~ Lamport introduced the following system of logical clocks. There is a clock C_i at each process P_i in the system C, can be thought as a function that assigns a no. $C_i(a)$ to any event a at P_i. The no. assign by system of clocks have no relation to physical time & hence the name logical clocks.

The logical clock takes monotonically increasing values these clocks can be implemented by counters.

The happen before relation ($\rightarrow$) can now be realized by using the logical clocks if the following two conditions are satisfied :-

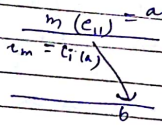
(i) For any two events a & b in a process P_i if a occurs before b then

P_i — a & b then $C_i(a) < C_i(b)$

If 'a' is event of sending a message 'm' in a process 'P_i' and 'b' is a event

Date: / /

of receiving same message 'm' at process P_j.



$c_i(a) < c_j(b)$

The following implementation rules for the clocks guarantee that the clock satisfies the correctness condition (i) & (ii)

i) clock c_i is implemented b/w any two successive events in process P_i.

$$c_i = c_i + d \quad (d > 0)$$

(ii) If a & b are two successive event in proces P_i and a happen before b $(a \to b)$ then

$$c_i(b) = c_i(a) + d$$

If event 'a' is the sending of message 'm' by process 'pi' then m is assigned a timestamp $(t_m) = c_i a$ on receiving the same message m by process P_j c_j is set to a value greater than or equal to

Date: / /

its present value and greater than t_m

$$c_j = \max(c_j, t_m + d)$$

→ 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 happed before b $(a \to b)$ then 'a' will also be positioned in that ordering before b.

$$c_i(a) < c_i(b).$$

However, lamport time stamp nothing can be said about the relationship b/w two events 'a' and 'b' by merely comparing their time values, Ca and Cb respectively.

In other words, if $Ca < Cb$ then this does not necessarily imply that a indeed happened before b.

Some more information is required for that

The problem with lamport timestamp is it do not capture causality This issue can be captured by means of vector time stamp.

Date: / /

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

Vector Timestamps are constructed by letting each process P_i maintain a vector V_i with the following two properties.

(i) $V_i[i]$ is no. of events that have occurred so far at P_i

(ii) If $V_i[j] = K$, then P_i is knowns as that K event has occured at P_j.

The first property is maintained by incrementing $V_i[i]$ at the occurance of each new event that happens at process P_i

The second property is maintained by piggy backing vector (every ~~prior~~ information) along with messages that are sent.

When P_i sends message m it also sends along its current vector as a time stamp V_T.

Date: / /

In this way a receiver is informed about the number of events that has occurred at an P_i.

However, is that the receiver is told ~~as~~ that how many events at other process have taken place before P_i sends message 'm'.

In other words, time stamp V_T of ~~m~~ message m tells the receiver how many events on other processes have proceeded 'm' and on which events 'm' may causally depends.

When process P_j receives message m it will adjust its own vector by setting each entry

$$\max(V_T(K), V_t[K])$$

The vector now reflex the no. of messages that P_j must receive to have atleast seen the same messages that proceeded the sending of m.

~~How after~~ Thereafter $V_J[J]$ is incremented by 1 representing the event of receiving a next message with a slight adjustment vector time stamp can be used to guarantee causal

Date: / /

message delivery.

In particular, a message 'r' is delieved only if

i) $V_t(r)[J] = V_k[j] + 1$

ii) $V_t(r)[i] \leq V_k[i]$ for all $i \neq j$

The first condition state that 'r' was the next message that P_k was expecting from process P_J.

The second condition states that P_k has not seen any message that was also not seen by P_J when it sends message 'r'. In particular then means that P_k has already seen message 'a'.

→ GLOBAL STATE

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

Date: / /

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 system have entered into deadlock state.

→ Casual Ordering of Messages

Election Algorithm

1) The Bully Algorithm

When any process notices that the coordinator is no longer responses to a ~~few~~ messages/ request it initiates an election. The process P holds on election as follows -

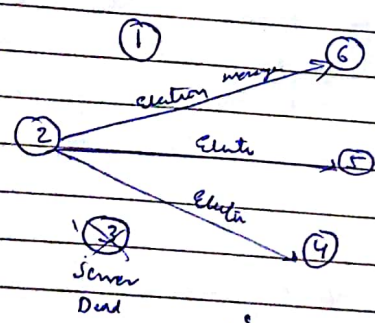
(i) P sends an election msg message to all the processes with higher ~~no~~ number if ~~no one~~ response with 'OK' message. P wins the election and becomes co-ordinator

(ii) If one of the higher up ~~answer~~ answers at

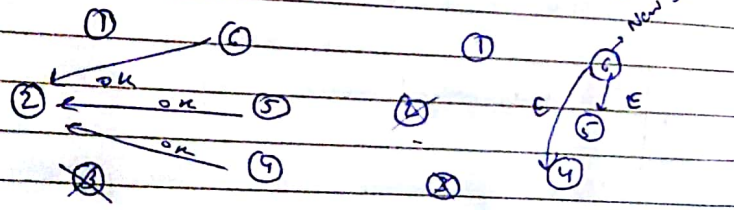
Date: / /

takes over and these jobs are done.

(2). Ring Algorithm



(i)



(ii)

→ Distributed Mutual Exclusion

→ Requirements of Mutual Exclusion Algorithm

The primary objective of a mutual exclusive algorithm is to maintain mutual exclusion that is to guarantee that only one request ~~passes~~ assess the critical section (C.S) at a time.

Date: / /

In addition following characteristics are considered important in an mutual Exclusive algorithm

1) Freedom from Deadlock.
2) Freedom from starvation.
3) Fairness
4) Fault Tolerance

1) Freedom from Deadlock :-

Two or more sites 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 repetiadly executing Critical section

3) Fairness :-

Fairness dictates that request must be executed in the order they were made Sense, the physical global clock does not exist time is determined by logical clock. Note that fairness implies freedom from starvation but not vice versa

Date: / /

4) Fault Tolerence :-

A mutual exclusion algorithm is fault tolerant if on the wake of a failure it can reorganize itself so that it continues to function without any dis. distraction

→ How to Measure the Performance ?

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

→ Token Based & Non - Token Based Algorithms.

1) Non - Token Based Algorithm :-

In Non - Token Based mutual Exclusion Algorithms a site communicates with a set of other sites to arbitrate (decide) who should execute the Critical Section next

For a site S_i, request set R_i contains ID's of all the site from which site S_i must acquire permission before entering into critical section.

Date: / /

Non - Token based Mutual Exclusion Algorithms use timestamps to order request for the critical section and to resolve the conflicts b/w simultaneous request for the critical section.

In these algorithms logical clocks are maintained and updated according to Lamport's scheme Each request for the critical section gets a timestamp & smaller timestamp request to have priority over larger timestamp request.

eg Lamport, Richart - Agrawala, Mackawa & Algorithms.

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 it possess the token.

Token based algorithm use 'sequence numbers' instead of timestamps. Every request for the token contains a sequence number and sequence number of the site advances independently

Date: / /

A site increment it sequence number counter every time it makes a request for the token

eg :- Suzuki - Ka Sami's Broadcast Algorithm and Singhal's heuristic, Raymond's tree

→ ~~Deadlock Detection in Distributed Systems~~

→ Non - Token Based :

- Lamport's Algorithm : Lamport was first one to give distributed ~~Algo~~ mutual exclusion algorithms as an illustration of its clock synchronisation scheme The lamport algorithm

$$\forall i : 1 \le i \le N :: R_i = \{s_1, s_2 \dots s_N\}$$

Every site s_i keeps a queue r_q^{ic} (Request Queue) which contains mutual exclusion request ordered by their timestamp

Algorithm

(a) Requesting the critical section

(i) when a site s_i wants to enter in a critical section it sends a request message

Date: / /

REQUEST (t_{s_i}, i) to all the sites in need its request site (R_i) and places the request on request queue where (t_{s_i}, i) is timestamp of the request.

ii) when a site s_j receives a request REQUEST (t_{s_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 site s_i enters in critical section when the following two conditions hold :-

i) s_j has received a message with timestamp larger than (t_{s_i}, i) all other site

ii) s_i request is at the top of request - queue

(c) Releasing the Critical Section

(v) Site s_i upon releasing the critical section removes its request from the top of its request queue and sends a timestamp release message to all the site in its request set

vi) when a site s_j receives the release message from site s_i it removes s_i request from its request queue

Date: / /

→ When a site removes a request from its request queue it owns 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

→ Token - Based Algorithm

(a) Requesting the Critical section

(i) If the requesting site S_i does not have token then it increments its sequence number $RN_i[i]$ and sends the ~~request~~ REQUEST (i, S_n) message to all the sites (S_n is the updated value of RN) "$RN_i[i]$"

(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 ideal 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

Date: / /

(c) Releasing the Critical Section

Having finish the execution of critical section site S_i takes the following options.

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

(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 site ID from the queue and sends the token to the site indicated by the ID.

⇒ Deadlock Detection on Distributed Systems

Deadlock can be dealt with using any one of the following three strategies

(i) Deadlock Prevention :-

It is commonly achieved by either having a process acquire all the needed resources simultaneously before it begins the execution. Or by pre-empting (pre-empting) the process that holds the needed resource

Date: / /

ii) Deadlock Avoidance :-

In this approach the distributed system resource is granted to a process if the resulting global system is safe

iii) 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 a deadlock process.