Name : Sabarivasan V
Register No : 205001085

UCS1701 - DISTRIBUTED SYSTEMS

DESIGN OF VECTOR CLOCKS FOR AD-HOC DISTRIBUTED SYSTEMS

Consider a modern ad-hoc distributed system like vehicular or mobile networks. As the characteristics of ad-hoc systems vary from the conventional distributed systems, the design challenges of ad-hoc systems should be studied for renovating the insight towards key technological shifts.
a. Adapt the conventional representation of vector clocks to suit the ad-hoc distributed system.
b. Formulate the implementation rules for the adapted design of vector clock synchronization concerning the design challenges of the ad-hoc distributed system.
c. Elaborate the working of the adapted design through a relevant example.

Discuss the answers in detail.
i. Highlight the abstract difference between a conventional distributed system and an ad hoc distributed system.
ii. Explain the reason that the vector clocks for conventional distributed systems are not suitable for ad-hoc distributed systems.
iii. Discuss the new design idea for vector clocks concerning the reasons mentioned for ii.
iv. Adapt the implementation rules for the new design discussed for iii.
v. Simulate the working of the newly designed vector clock through relevant and detailed examples for various cases depicting different degrees of ad-hocness

Difference between a conventional distributed system and an ad hoc distributed system.

Conventional Distributed Systems	**Ad Hoc Distributed Systems**
These systems usually have a predefined, static network topology. They follow a structured, often hierarchical or centralized, architecture. Nodes and resources are typically known in advance, and configurations are carefully managed.	Ad hoc systems have a dynamic and self-organizing network topology. Nodes in these systems are not necessarily known in advance and can join or leave the network at any time. The configuration and structure of the network can change rapidly and autonomously.
Resource management is typically centralized and planned. Resources are allocated based on predefined policies and often involve centralized ntrol mechanisms.	Resource management is more decentralized. Nodes in an ad hoc system may need to self-configure and manage resources autonomously without central coordination.
Maintaining a consistent state is easier using snapshots.	Owing to the dynamic nature of the system, maintaining consistency is difficult.
Fault tolerance and isolation is easy.	Identifying faults and mitigating them can prove to be cumbersome.
These systems have a clear, stated goal in mind when they are created. Usually, they are preconfigured to carry out a certain function, and the infrastructure is meticulously designed and arranged. Client-server networks, cloud computing environments, and data centers are a few examples.	These systems are intended for situations that are more spontaneous and dynamic. They are designed to be quickly and easily assembled, frequently in circumstances where a prearranged infrastructure is impractical or unavailable. Ad hoc systems find frequent application in disaster recovery, military networks, and impromptu teamwork.

Explain the reason that the vector clocks for conventional distributed systems are not suitable for ad-hoc distributed systems.

Vector clocks and ad hoc distributed systems have characteristics that make them less suitable for each other. Vector clocks are more appropriate for conventional distributed systems, while ad hoc systems may benefit from other approaches for tracking causality and ordering events. Here are some reasons why vector clocks may not be ideal for ad hoc distributed systems:

Centralization and Predictability:
Vector clocks often rely on centralized or predetermined knowledge of the participants in a distributed system. In ad hoc systems, the participants may join and leave dynamically, making it challenging to maintain a central knowledge of all nodes and their relationships. Ad hoc systems are inherently decentralized and less predictable.

Scalability:
Vector clocks can become complex and inefficient as the number of participants in the system increases. In ad hoc systems, the number of participants can grow rapidly and unpredictably, making it challenging to manage vector clocks for all nodes in a scalable manner.

Overhead:
Maintaining vector clocks for every node and updating them with each event can introduce significant overhead in terms of message size and processing. In ad hoc systems, where resources and bandwidth may be limited, this overhead can be prohibitive.

Dynamic Network Topology:
Ad hoc systems often have a dynamic and self-organizing network topology. Nodes may join, leave, or move within the network frequently. Vector clocks rely on a stable set of participants, which may not align with the ever-changing nature of ad hoc networks.

Complexity of Causality Tracking:
Vector clocks are used to track causality between events in a distributed system. In ad hoc systems, determining causality can be particularly challenging due to the dynamic and unpredictable nature of the network. Vector clocks may not provide a clear picture of causality in such scenarios.

Resource Constraints:
Ad hoc systems are often designed for resource-constrained environments, such as mobile networks or IoT devices. Vector clocks can be resource-intensive, making them less suitable for these environments.

Adaptability:
Ad hoc systems need to be highly adaptable to rapidly changing conditions. Vector clocks, designed for more stable and controlled environments, may not provide the adaptability needed to handle the dynamic nature of ad hoc systems.

New design idea for vector clocks

Dynamic Vector:
The number of processes in the system is not constant. So the vector cannot have a fixed size. Maintain a counter variable and when a node joins the system, increment the counter variable and store the clock value and process ID. When a process leaves, decrement the counter,
remove the clock value from the vector and shift all the values in the vector.

Vector Clock Updates:
When messages are exchanged, update the vector clock using the timestamp.

Stagnant Process:
Maintain a TTL value which when expired removes the corresponding process's entry from the vector clock. If any message exchange takes place involving that process, the vector clock entry would be updated.

An alternative method would be to make a new node or a leaving node broadcast a message advertising its entry or departure.

Causality:
Maintain a time window. The processes occurring within that window are considered concurrent and not causal.

Renewing Process Clock Value:
When timestamp is passed, comparison has to be done with the values in the timestamp and the values in the vector clock of the receiver
process. This helps to check if outdated processes are present in the vector clock. If any process's entry has expired, but is present in
either the timestamp or the vector clock entry, it will get renewed.

Implementation rules for the new design

All the processes should maintain a queue initially with its own identifier in the queue and clock value. Vector is an array of
structures and for process i will look like [{i, 0, t, 1}] C_i.id = i,
C_i.clock = 0 and C_i.ttl = t, C_i.counter = 1

For process i, C_i.ttl = ∞

Sending Event/All events:
C_i.id == i -> C_i.clock++
C_k.ttl--

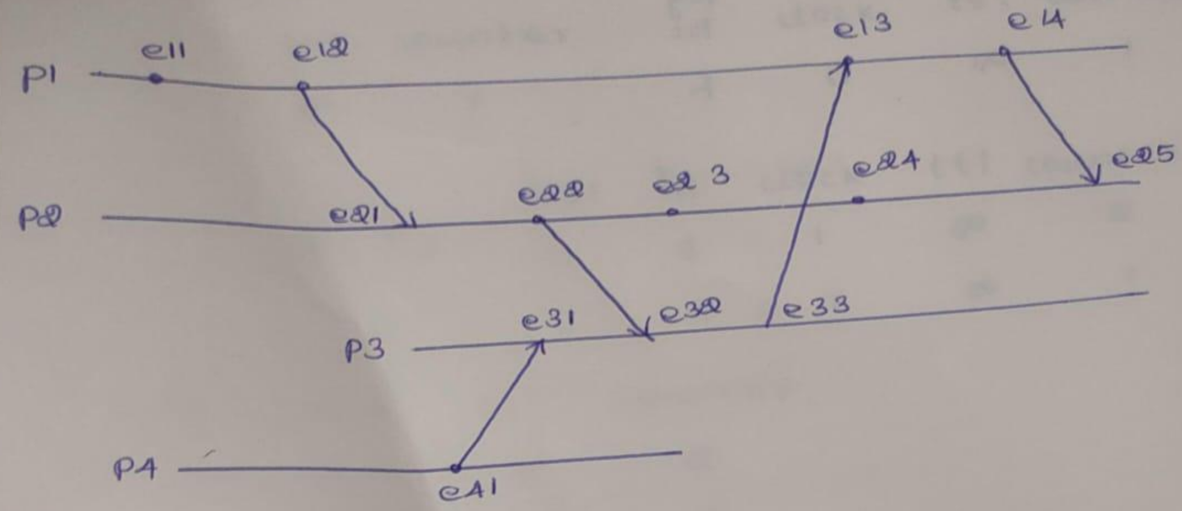
Receiving Event:
Sender -> i
Receiver -> j
Other Processes -> k
Time stamp sent -> tm

```
If process id is not in the vector clock of j, then
        ++counter
        $C_j$[counter].id = k
        $C_j$[counter].clock = Clock of value of that process in the time stamp
Else                                                $C_j$.id == k
                                                    $C_j$.clock = max($C_j$.clock, tm.clock)
```

Expired Process:
If C_i.id == k and C_i.ttl == 0, remove it from the queue, C_i.counter--.

Working of the newly designed vector clock

Let t = 3.



vector table updates

Initial Value

P1

id	clock	ttl	counter
1	0	∞	1

P2

id	clock	ttl	counter
1	0	∞	1

After e11

P1 [1, 1, ∞, 1]

After e12

P1 [1, 2, ∞, 1]

After e21

P2

id	clock	ttl	counter
2	1	∞	2
1	2	3	1

Initial tables for P3. P1

p3

id	clock	ttl	counter
3	0	∞	1

P1

id	clock	ttl	counter
1	0	∞	1

e11

P1 [1. 1. ∞. 1]

e31

id	clock	ttl	counter
3	1	∞	2
1	1	∞	1

e22

p2	id	clock	ttl	counter
	2	2	∞	2
	1	2	2	1

e32

p3	id	clock	ttl	counter
	3	2	∞	3
	2	2	3	2
	1	2	2	1

e23

p2	id	clock	ttl	counter
	2	3	∞	2
	1	2	1	1

e33

p3	id	clock	ttl	counter
	3	3	∞	3
	2	2	2	2
	1	2	1	1

e13

P1	id	clock	ttl	counter
	1	3	∞	3
	2	2	2	2
	3	3	3	3