UNIT – 02

DISTRIBUTED DEADLOCKS

UNIT-02/LECTURE-01

Deadlock ([RGPV/ Dec 2011 (5)]

In an operating system, a deadlock is a situation which occurs when a process or thread enters a waiting state because a resource requested is being held by another waiting process, which in turn is waiting for another resource. If a process is unable to change its state indefinitely because the resources requested by it are being used by another waiting process, then the system is said to be in a deadlock.

Deadlock is a common problem in multiprocessing systems, parallel computing and distributed systems, where software and hardware locks are used to handle shared resources and implement process synchronization.

Necessary condition for deadlock ([RGPV/ Dec 2011 (5)]

A deadlockers situation can arise if all of the following conditions hold simultaneously in a system:

1. Mutual Exclusion: At least one resource must be held in a non-shareable mode. Only one process can use the resource at any given instant of time.
2. Hold and Wait or Resource Holding: A process is currently holding at least one resource and requesting additional resources which are being held by other processes.
3. No Preemption: a resource can be released only voluntarily by the process holding it, after that process has completed its task
4. Circular Wait: A process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource. In general, there is a set of waiting processes, P = {P1, P2, ..., PN}, such that P1 is waiting

for a resource held by P2, P2 is waiting for a resource held by P3 and so on until PN is waiting for a resource held by P1.

Distributed Deadlocks

A distributed system consists of a number of sites connected by a network. Each site maintains some of the resources of the system. Processes with a globally unique identifier run on the distributed system. They make resource requests to a controller. There is one controller per site. If the resource is local, the process makes a request of the local controller. If the desired resource is at a remote site, the process sends a message. After a process makes a request, but before it is granted, it is blocked and said to be dependent on the process that holds the desired resource.

The controller at each site could maintain a WFG on the process requests that it knows about. This is the local WFG. However, each site's WFG could be cycle free and yet the distributed system could be deadlocked. This is called global deadlock. This would occur in the following situation:

1. Process A at site 1 holds a lock on resource X.
2. Process A has requested, but has not been granted, resource Y at site 2.
3. Process B at site 2 holds a lock on resource Y.
4. Process B has requested, but has not been granted, resource X at site 1.

Both processes are blocked by the other one. There is a global deadlock. However, the deadlock will not be discovered at either site unless they exchange information via some detection scheme.

Distributed Detection Schemes

A detection scheme is evaluated by two criteria: 1) If there exists an actual deadlock, it must be detected in a finite amount of time, and; 2) The scheme must not find a deadlock that is

not actually there.

One way to detect deadlocks in distributed systems is for each site to construct a local WFG on the part it knows about. A site that suspects deadlock initiates a global snapshot protocol that constructs a consistent global state of the distributed system (see Distributed Snapshots). The global state corresponds to a deadlock if the union of the local WFGs has a cycle. This algorithm is correct because deadlock is a stable property.

Communication Deadlocks

A communication deadlock occurs, for example, when process A is trying to send a message to process B, which in turn is trying to send one to process C, which is trying to send one to A.

S.NO	RGPV QUESTIONS	Year	Marks
Q.1	What is deadlock? What are the necessary condition for deadlock.	Dec 2011	5
Q.2	Describe various deadlock handling technique.	Dec 2013	5

UNIT-02/LECTURE-02

Resource Deadlocks ([RGPV/ Dec 2013 (3.5)]

A resource deadlock occurs when processes are fighting over exclusive access to I/O devices, files, locks, or other resources.

• Strategies are used to handle deadlocks

1.The ostrich algorithm (ignore the problem)

2.Detection (let deadlocks occur, detect them, and try to recover)

3.Prevention (statically make deadlocks structurally impossible)

4.Avoidance (avoid deadlocks by allocating resources carefully)

Deadlock prevention ([RGPV/ Dec 2013(7)]

Deadlock prevention is commonly achieved by either having a process acquire all the needed resources simultaneously before it begins execution or by preempting a process that holds the needed resource. In the former method, a process requests (or releases) a remote resource by sending a request message (or release message) to the site where the resource is located. This method has a number of drawbacks. First, it is inefficient as it decreases the system concurrency. Second, a set of processes can become deadlocked in the resource acquiring phase. For example, suppose process PI at site *SI* and process P2 at site *S2* simultaneously request two resources *R3* and 14, located at sites *S3* and *S4,* respectively. It may happen that *S3* grants *R3* to PI and *S4* grants *R4* to *P2,* resulting in a deadlock. This problem can be handled by forcing processes to acquire needed resources one by one; however, this approach is highly inefficient and impractical. Third, in many systems, future resource requirements are unpredictable (Le., not known a priori).

Distributed Deadlock Prevention

Wait-die-An old process wants a resource held by a young process. A young process wants a

resource held by an old process. In one case we should allow the process to wait; in the other we should kill it. e.g. if (a) dies and (b) wait. Then killing off an old process trying to use a resource held by a young process, which is inefficient. This algorithm is called **wait-die.**

Wound wait Algorithm

One transaction is supposedly wounded (it is actually killed) and the other waits. If an old process wants a resource held by a young process the old process preempts the young one whose transaction is then killed. The young one probably starts up again immediately & tries to acquire the resource forcing it to wait.

S.NO	RGPV QUESTIONS	Year	Marks
Q.1	Explain the following: distributed resource deadlock.	Dec 2013	3.5
Q.2	What are the methods to prevent distributed deadlock?	Dec 2013	7

UNIT-02/LECTURE-03

Deadlock Avoidance ([RGPV/ June 2014 (7)]

In the deadlock avoidance approach to distributed systems, a resource is granted to a process if the resulting global system state is safe (note that a global state includes all the processes and resources of the distributed system), Because of the following problems, deadlock avoidance can be seen as impractical in distributed systems:

(1) Every site has to maintain information on the global state of the system, which translates into huge storage requirements and extensive communication costs (as every change in the global state has to be communicated to every site),

(2) The process of checking for a safe global state must be mutually exclusive, because if several sites concurrently perform checks for a safe global state (each site for a different resource request), they may all find the state safe but the net global state may not be safe. This restriction will severely limit the concurrency and throughput of the system. (3) Due to the large number of processes and resources, it will be computationally expensive to check for a safe state.

Distributed Deadlock Detection ([RGPV/ June 2013 (7)], ([RGPV/ June 2014 (7)]

A distributed system is a network of sites that exchange information with each other by message passing. A site consists of computing and storage facilities and interface to local users and a communication network. In distributed systems, a process can request and release resources (local or remote) in any order, which may not be known a priori and a process can request some resources while holding others. If the sequence of the allocation of resources to processes is not controlled in such environments, deadlocks can occur.

Deadlock detection requires an examination of the status of process-resource interactions for the presence of cyclical wait. Deadlock detection in distributed systems has two favorable conditions:

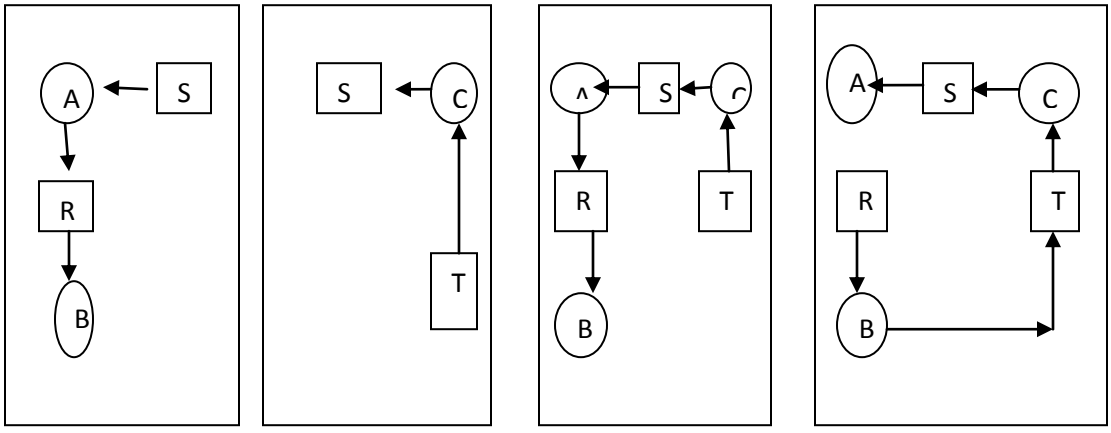
(1) Once a cycle is formed in the WFG, it persists until it is detected and broken and

(2) cycle detection can proceed concurrently with the normal activities of a system (and therefore it does not have a negative effect on system throughput). Because of this, the literature on deadlock handling in distributed systems is highly focused toward deadlock detection methods. In this chapter, the discussion is limited to deadlock detection techniques in distributed systems.

Centralized Deadlock Detection ([RGPV/ June 2013 (7)], ([RGPV/ June 2014 (7)]

Centralized Deadlock Detection - Each machine maintains the resource graph for its own processes and resources, a central coordinator maintains the resource graph for the entire system (the union of all the individual graphs). When the coordinator detects a cycle, it kills off one process to break the deadlock.

False Deadlock- Detecting a nonexistent deadlock in distributed systems has been referred to as false deadlock detection. False deadlock wi1l never occur in a system of two-phase locking transactions & the coordinator conclude incorrectly that a deadlock exist and kills some process.



(a) (b) (c) (d)

(a) Initial resource graph for machine 0(b) Initial resource graph for machine 1

(c) The coordinator's view of the world. (d) The situation after the delayed message.

Distributed Deadlock Detection- a special **probe** message is generated and sent to the process (or processes) holding the needed resources. The message consists of three numbers: the process that just blocked, the process sending the message, and the process to whom it is being sent. The initial message from 0 to 1 contains the triple (0, 0, 1).

The Probe Scheme

The scheme proposed by Chandy, Misra and Haas [1] uses local WFGs to detect local deadlocks and probes to determine the existence of global deadlocks. If the controller at a site suspects that process A, for which it is responsible, is involved in a deadlock it will send a probe message to each process that A is dependent on. When A requested the remote resource, a process or agent was created at the remote site to act on behalf of process A to obtain the resource. This agent receives the probe message and determines the dependencies at the current site.

The probe message identifies process A and the path it has been sent along. Thus, the message probe(i,j,k) says that it was initiated on behalf of process i and was sent from the controller for agent j (which i is dependent on) to the controller for agent k.

When an agent whose process is not blocked receives a probe, it discards the probe. It is not blocked so there is no deadlock. An agent that is blocked sends a probe to each agent that it is blocked by. If process i ever receives probe(i,j,i), it knows it is deadlocked.

This popular approach, often called "edge-chasing", has been shown correct in that it detects all deadlocks and does not find deadlocks when none exist.

In the OR model, where only one out of several requested resources must be acquired, all of the blocking processes are sent a query message and all of the messages must return to the originator in order for a deadlock to be declared. The query message is similar to a probe, but has an additional identifier so the originator can determine whether or not all the paths were covered.

S.NO	RGPV QUESTIONS	Year	Marks
Q.2	What are the limitations of centralised deadlock detection?	June 2013	7
Q.3	Differentiate between centralized and distributed deadlock detection.	June 2014	7
Q.4	Explain the following: deadlock avoidance.	June 2014	7

UNIT-02/LECTURE-04

Obermarck's Path-Pushing Algorithm ([RGPV/ Dec 2011 (7)] .([RGPV/ June 2014 (3.5)]

Individual Sites maintain local WFG

A virtual node 'x' exists at each site

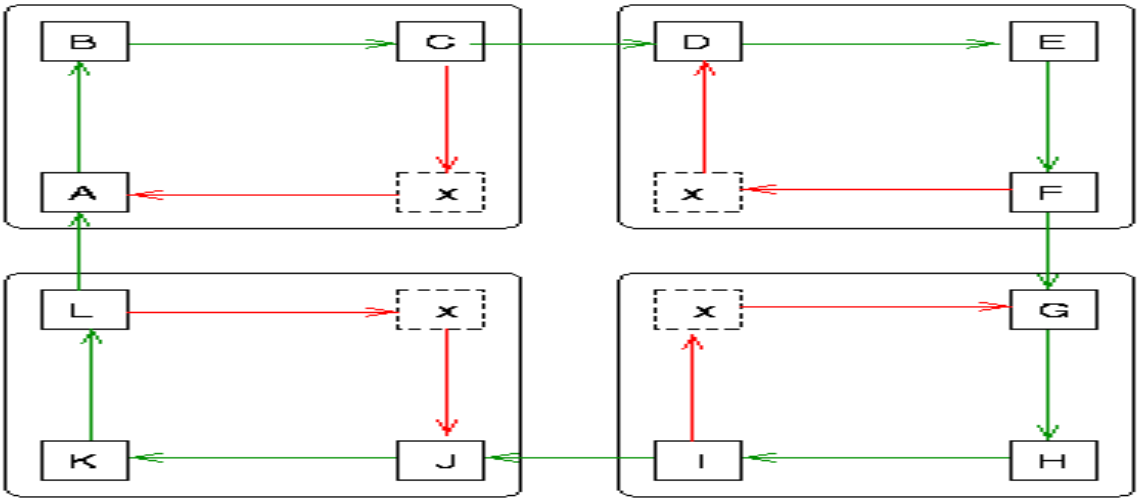
Node 'x' represents external processes

Detection Process

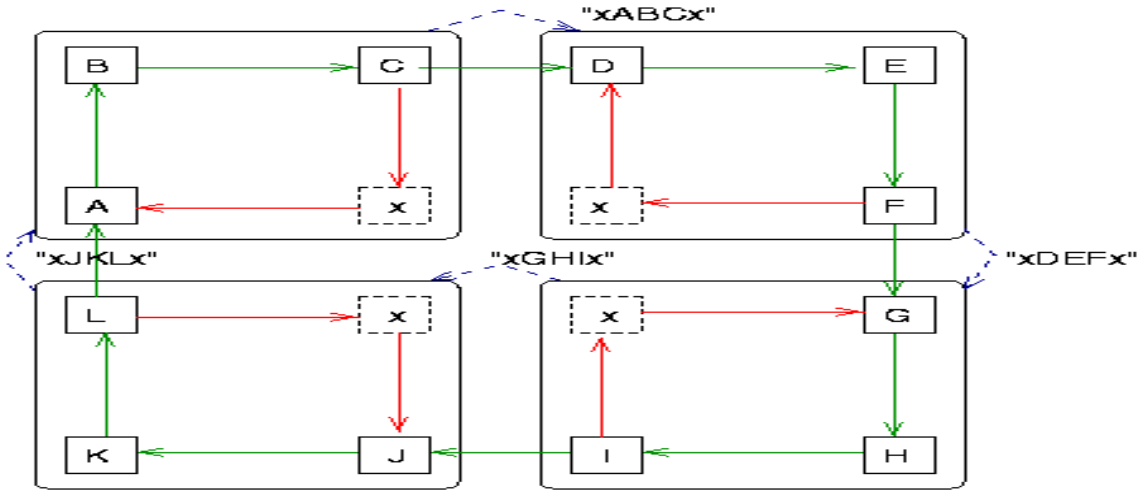
- Case 1: If Site S_n finds a cycle not involving 'x' -> Deadlock exists.
- Case 2: If Site S_n finds a cycle involving 'x' -> Deadlock possible.
- If Case 2 ->
- Site S_n sends a message containing its detected cycles to other sites. All sites receive the message, update their WFG and re-evaluate the graph.
- Consider Site S_j receives the message:
- Site S_j checks for local cycles. If cycle found not involving 'x' (of S_j) -> Deadlock exists.
- If site S_j finds cycle involving 'x' it forwards the message to other sites.
- Process continues till deadlock found.

Obermark's Algorithm Example

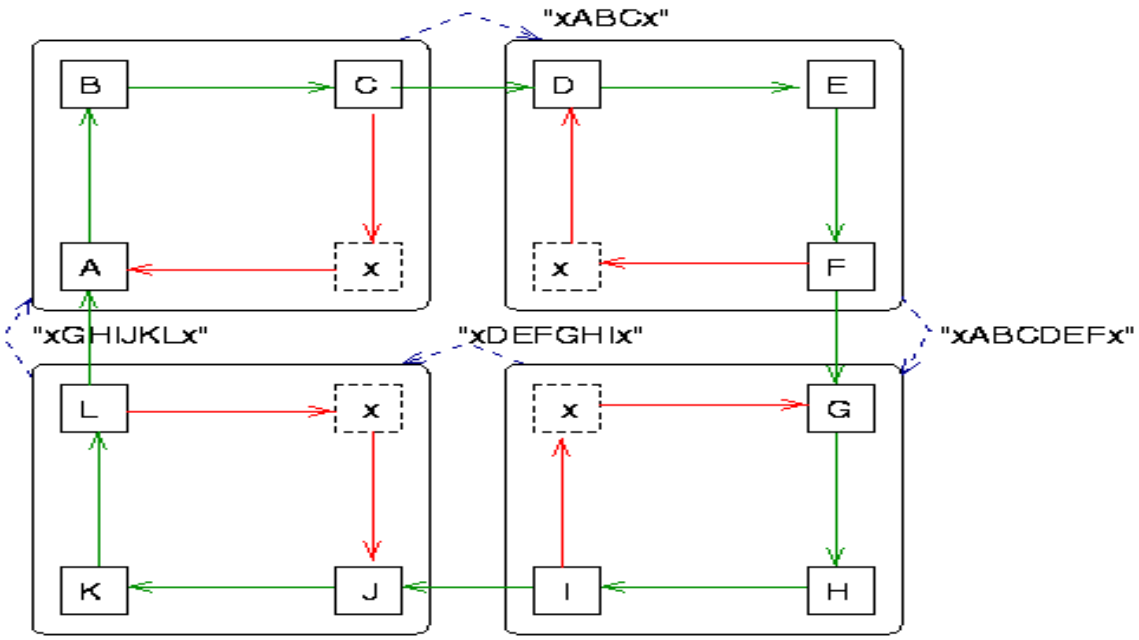
Intial State



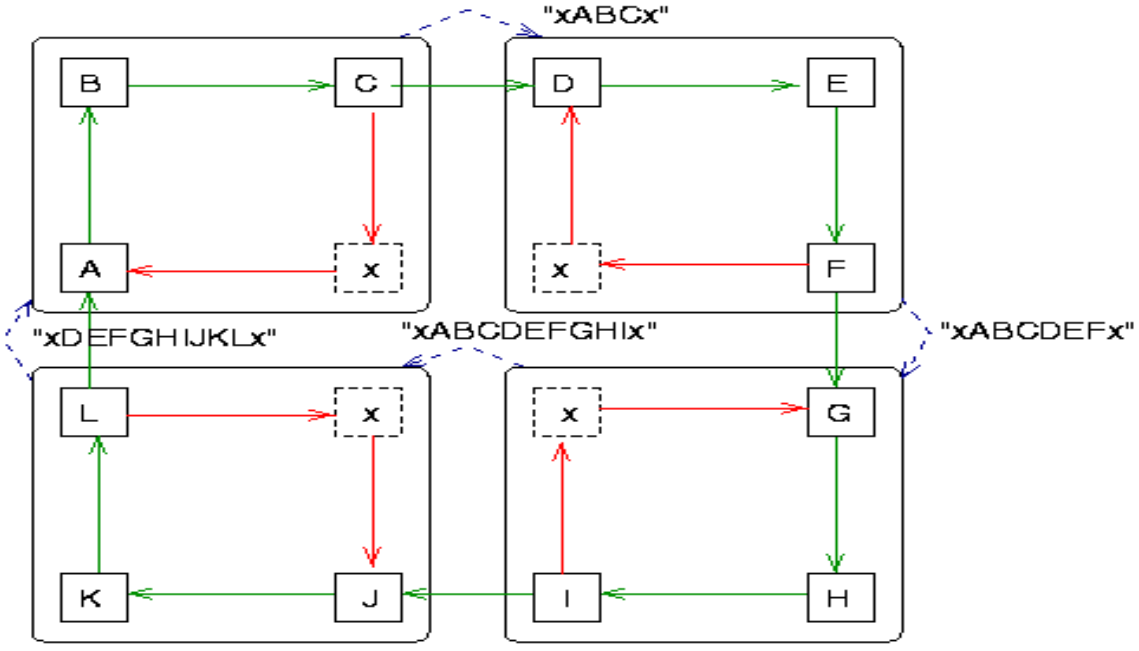
Iteration 1



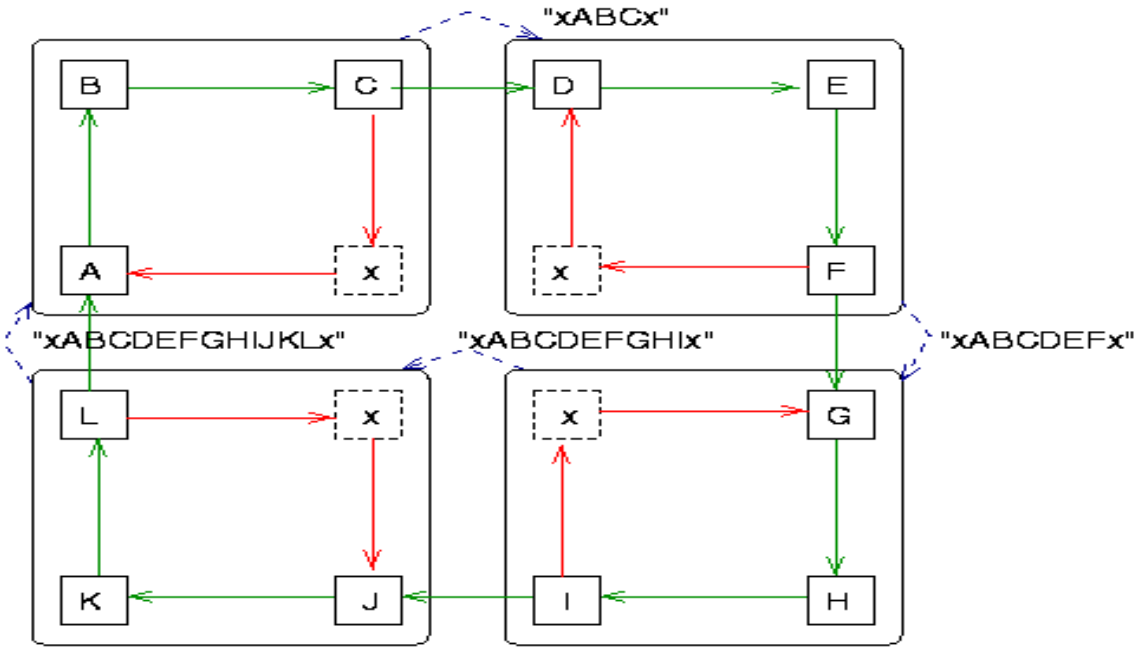
Iteration 2



Iteration 3



Iteration 4



S.NO	RGPV QUESTIONS	Year	Marks
Q.1	Explain the following: Path pushing algorithm	June 2014	3.5
Q.2	What are the different deadlock handling strategies? Explain a distributed deadlock detection algorithm.	Dec 2011	7

UNIT-02/LECTURE-05

Edge chasing algorithms ([RGPV/ June 2013 (5)] ([RGPV/ June 2014 (5)]

Chandy-Misra-Haas's Algorithm (AND MODEL):

- A probe(i, j, k) is used by a deadlock detection process Pi. This probe is sent by the home site of Pj to Pk.
  - This probe message is circulated via the edges of the graph. Probe returning to Pi implies deadlock detection.
  - Terms used:
    - Pj is dependent on Pk, if a sequence of Pj, Pi1,.., Pim, Pk exists.
    - Pj is locally dependent on Pk, if above condition + Pj,Pk on same site.
    - Each process maintains an array dependenti: dependenti(j) is true if Pi knows that Pj is dependent on it. (initially set to false for all i & j).

Sending the probe:

if Pi is locally dependent on itself then deadlock.

else for all Pj and Pk such that

(a) Pi is locally dependent upon Pj, and

(b) Pj is waiting on Pk, and

(c) Pj and Pk are on different sites, send probe(i,j,k) to the home site of Pk.

Receiving the probe:

if (d) Pk is blocked, and

(e) *dependentk(i)* is false, and

(f) Pk has not replied to all requests of Pj,

then begin

dependentk(i) := true;

if k = i then Pi is deadlocked

else ...

Receiving the probe:

else for all Pm and Pn such that

(a') Pk is locally dependent upon Pm, and

(b') Pm is waiting on Pn, and

(c') Pm and Pn are on different sites, send probe(i,m,n)

to the home site of Pn.

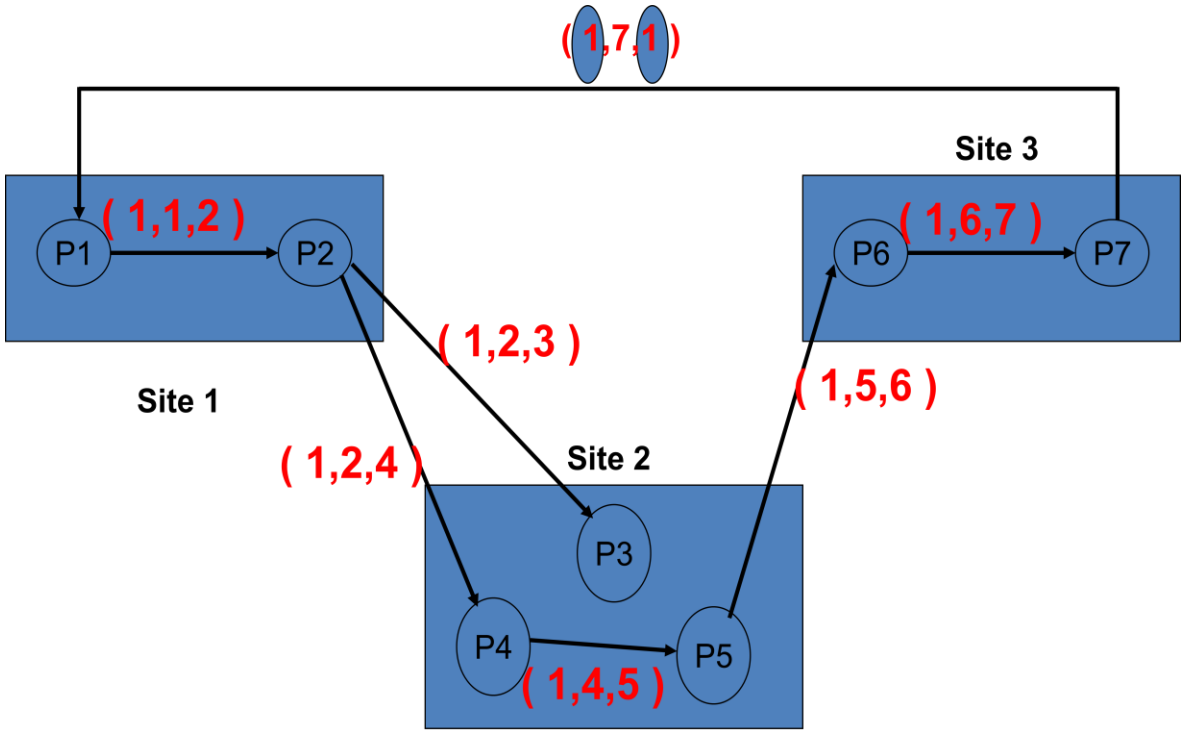
end.

Performance

For a deadlock that spans m processes over n sites, m(n-1)/2 messages are needed.

Size of the message 3 words.

Delay in deadlock detection O(n).



S.NO	RGPV QUESTIONS	Year	Marks
Q.1	Explain the Edge chasing algorithm.	June 2014,June 2013	5

UNIT -02/LECTURE -06

Agreement Protocols: Introduction

- Processes/Sites in distributed systems often compete as well as cooperate to achieve a common goal.
- Mutual Trust/agreement is very much required.
- In Distributed Data bases, there may be a situation where data managers have to decide “Whether to commit or Abort the Transaction”
- When there is no failure, reaching an agreement is easy.
- However, in case of failures, processes must exchange their values with other processes and relay the values received from others several times to isolate the effect of faulty processor.
- Agreement Protocols helps to reach an agreement in presence of failures.

System Model

- Agreement Problems have been studied under following System Model:

1. ‘n’ processors and at most ‘m’ of the processors can be faulty
2. Processors can directly communicate with other processors by message passing.
3. Receiver knows the identity of the sender
4. Communication medium is reliable.
5. Only Processors are prone to failures.

Synchronous VS Asynchronous Computation ([RGPV/ June 2011 (7)]

- Synchronous Computation

1. Processes run in lock step manner[Process receives a message sent to it earlier, performs computation and sends a message to other process.
2. Step of Synchronous computation is called round

- Asynchronous Computation

1. Computation does not proceed in lock step.

2. Process can send receive messages and perform computation at any time.

Model of processor Failures

- Processor Can Fail in three modes:

1. Crash Fault :

   (i) Processor stops and never resumes operation.

2. Omission Fault :

   (i) Processor Omits to send message to some processors

3. Malicious Fault :

(i) Also known as Byzantine Faults

(ii) Processor may send fictitious values/message to other processes to confuse them

(iii) Tough to detect/correct

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain Synchronous VS Asynchronous Computation.	June 2011	7

UNIT -02/LECTURE -07

Authenticated VS Non-Authenticated Messages

- Authenticated Messages:

1. Also known as signed Message
2. Processor cannot forge/change a received message
3. Processor can verify the authenticity of the message
4. It is easier to reach on an agreement in this case

- Non-Authenticated Messages:

1. Also known as Oral Message
2. Processor can forge/change a received message and claims to have received it from others
3. Processor cannot verify the authenticity of the message in this case.

Performance Aspects of Agreement Protocols

- Following Metrics are used :

1. Time: No of rounds needed to reach an agreement
2. Message Traffic: Number of messages exchanged to reach an agreement.
3. Storage Overhead: Amount of information that needs to stored at processors during execution of the protocol.

Classification of Agreement Problems

- There are three well known agreement problems in distributed systems:

1. Byzantine Agreement Problem :

- A single Value is to be agreed upon.
- Agreed Value is initialized by an arbitrary processor and all non faulty processors have to agree on that value.

2. Consensus Problem:

- Every processor has its own initial value and all non faulty processors must agree on a single common value.

3. Interactive Consistency Problem:

- Every processor has its own initial value and all non faulty processors must agree on a set of common values.
- In All the previous mentioned problems, all non faulty processors must reach an agreement
- In Byzantine and Consensus problems , agreement is on a single value
- In Interactive Consistency problem, agreement is on a set of common values.
- In Byzantine agreement problem, only one processor initializes the value where as in other two cases, every processor has its own initial value.

Byzantine Agreement Problem ([RGPV/ June 2014 (7)]

- Source Processor [Any arbitrarily chosen processor] broadcasts its values to others.
- Solution must meet following objectives:

1. Agreement: All non-faulty processors agree on the same value.
2. Validity: If source is nonfaulty, then the common agreed value must be the value supplied by the source processor.

"If source is faulty then all non- faulty processors can agree on any common value".

"Value agreed upon by faulty processors is irrelevant".

Model :

- Total of n processes, at most m of which can be faulty
- Reliable communication medium
- Fully connected
- Receiver always knows the identity of the sender of a message
- Byzantine faults
- Synchronous system

• In each round, a process receives messages, performs computation, and sends messages.

Byzantine Agreement

Also known as Byzantine Generals problem

– One process x broadcasts a value v

• Agreement Condition: All non non-faulty processes must agree on a common value.

• Validity Condition: The agreed upon value must be v if x is non non-faulty.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Show that the Byzantine Agreement cannot always be reached among four processors if two processor are faulty.	June 2014	7

UNIT -02/LECTURE -08

Consensus Problem ([RGPV/ June 2013 (7)], ([RGPV/ June 2014 (7)]

- Every Processor broadcasts its initial value to all other processors.
- Initial Values may be different for different processors.
- Protocol must meet following objectives:

1. Agreement : All non-faulty processors agree on the same single value.
2. Validity : If initial value of every non-faulty processor is v , then the common agreed value by all non-faulty processors must be v.

"If initial value of non-faulty processors are different then all non- faulty processors can agree on any common value".

"Value agreed upon by faulty processors is irrelevant".

Interactive Consistency Problem

- Every Processor broadcasts its initial value to all other processors.
- Initial Values may be different for different processors.
- Protocol must meet following objectives:

1. Agreement : All non-faulty processors agree on the same vector (v1,v2,…vn).
2. Validity : If ith processor is non-faulty and its initial value is vi, then the ith value agreed by all non-faulty processors must be vi.

"If jth processor is faulty then all non- faulty processors can agree on any common value vj".

"Value agreed upon by faulty processors is irrelevant".

Solution for Byzantine Agreement Problem

- First Defined and solved by lamport.
- Source Broadcasts its initial value to all other processors.
- Processors send their values to other processors and also relay received values to others.

- During Execution faulty processors may confuse by sending conflicting values.
- However if faulty processors dominate in number, they can prevent non-faulty processors from reaching an agreement.
- No of faulty processors should not exceed certain limit.
- Pease showed that in a fully connected network, it is impossible to reach an agreement if number faulty processors 'm' exceeds (n-1)/3. n = number of processors

Applications of Agreement Algorithms

1. Fault-Tolerant Clock Synchronization

- Distributed Systems require physical clocks to synchronized
- Physical clocks have drift problem
- Agreement Protocols may help to reach a common clock value.

2. Atomic Commit in DDBS

- DDBS sites must agree whether to commit or abort the transaction
- Agreement protocols may help to reach a consensus.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Write on classification of agreement problem and in what way consensus problem is different than other problems in distributed systems?	June 2013	7
Q.2	Explain the following: Consensus Problem	June 2014	7

UNIT -02/LECTURE- 09

System Model

Conditions for deadlocks and methods to handle deadlocks in various special cases were discussed. The problem of deadlocks has been generally studied in distributed systems under the following model:

The systems have only reusable resources.

•Processes are allowed only exclusive access to resources.

•There is only one copy of each resource.

A process can be in two states: running or blocked. In the running state (also called the active state), a process has all the needed resources and is either executing or is ready for execution. In the blocked state, a process is waiting to acquire some resources.

Resource Vs Communication deadlocks

Two types of deadlocks have been discussed in the literature: resource deadlock and communication deadlock. In resource deadlocks, processes can simultaneously wait for several resources and cannot proceed until they have acquired all those resources. A set of processes is resource-deadlocked if each process in the set requests resources held by another process in the set and it must receive all of the requested resources before it can become unblocked.

In communication deadlocks ,processes wait to communicate with other process among a set of processes. A waiting process can unblock on receiving a communication from anyone of these processes. A set of processes is communication-deadlocked if each process in the set is waiting to communicate with another process in the set and no process in the set ever initiates any further communication until it receives the communication for which it is waiting. Note that a "wait to communicate" can be viewed as a "wait to acquire a resource".

Persistence & Resolution

1.Deadlock persistence:

a. Average time a deadlock exists before it is resolved.

2.Implication of persistence:

a. Resources unavailable for this period: affects utilization

b. Processes wait for this period unproductively: affects response time.

3.Deadlock resolution:

a. Aborting at least one process/request involved in the deadlock.

b. Efficient resolution of deadlock requires knowledge of all processes and resources.

c. If every process detects a deadlock and tries to resolve it independently -> highly inefficient ! Several processes might be aborted.

4. Priorities for processes/transactions can be useful for resolution.

a. Consider priorities introduced in Obermarck's algorithm.

b. Highest priority process initiates and detects deadlock (initiations by lower priority ones are suppressed).

c. When deadlock is detected, lowest priority process(es) can be aborted to resolve the deadlock.

5.After identifying the processes/requests to be aborted,

a. All resources held by the victims must be released. State of released resources restored to previous states. Released resources granted to deadlocked processes.

b. All deadlock detection information concerning the victims must be removed at all the sites.

REFERENCES

BOOK	AUTHOR	PRIORITY
Distributed operating systems; Concepts and design	P K Sinha	1
Distributed systems: Principles and paradigms	Tanenbaum and steen	2