International Journal of
Emerging Research in Management &Technology
ISSN: 2278-9359 (Volume-4, Issue-12)

Research Article | **December 2015**

Approaches for Deadlock Detection for Distributed Systems

Prabhsimran Singh
P.G. Deptt. of Computer Science
Khalsa College, India

Sukhmanjit Kaur
Department of Computer Science
Guru Nanak Dev University, India

Neha Bassan
Department of Computer Science
Bhutta Group of Colleges, India

Abstract—

In today environment Distributed database is mainly used by large organization for their striking features. When we develop a deadlock detection and prevention approaches for distributed database. A deadlock is a condition in a system where a process cannot proceed because it needs to obtain a resource held by another process but it itself is holding a resource that the other process needs. The same conditions for deadlocks in uniprocessors apply to distributed systems. Unfortunately, as in many other aspects of distributed systems, they are harder to detect, avoid, and prevent. Deadlocks are a fundamental problem in distributed systems. Deadlock detection is more difficult in systems where there is no such central agent and processes may communicate directly with one another. Deadlock detection and resolution is one among the major challenges faced by a Distributed System. In this paper, we discuss deadlock detection techniques and present approaches for detecting deadlocks in Distributed Systems. We wish that our paper had served as a survey of the important solutions in the fields of deadlock for distributed system.

Keywords— Deadlocks, Distributed Systems, Mutual Exclusion.

I. INTRODUCTION

Deadlock detection is more difficult in systems where there is no such central agent and processes may communicate directly with one another. In this paper, we discuss deadlock detection techniques and present approaches for detecting deadlocks in Distributed Systems.

What are Deadlocks?

A deadlock[1,2] is a state where a set of processes request resources that are held by other processes in the set. A deadlock is a condition in a system where a process cannot proceed because it needs to obtain a resource held by another process but it itself is holding a resource that the other process needs. Fig 1. Shows Resource R1 is requested by Process P1 but is held by Process P2.

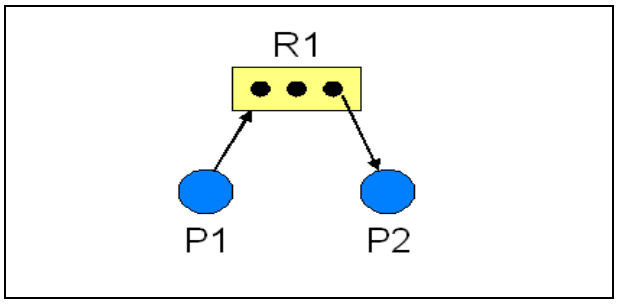

Figure 1. Resource R1 is requested by Process P1 but is held by Process P2

- **The various causes of Deadlock are as following:**
  a. **Mutual Exclusion** – Resources being held must be in non-shareable mode.
  b. **Hold n Wait –** A Process is holding one resource and is waiting for another, which is held by another process.
  c. **No Preemption –** Resource cannot be preempted even if it is being requested
  d. **Circular Wait –** Presence of a cycle of waiting processes.

- **Distributed Database System:**

Distributed database management system[3] is software for managing databases stored on multiple computers in a network. A distributed database is a set of databases stored on multiple computers that typically appears to applications on a single database. Consequently, an application can simultaneously access and modify the data in several databases in a network. DDBMS is specially developed for heterogeneous database platforms, focusing mainly on heterogeneous database management systems (HDBMS).

A database physically stored in two or more computer systems. Although geographically dispersed, a distributed database system manages and controls the entire database as a single collection of data. If redundant data are stored in separate databases due to performance requirements, updates to one set of data will automatically update the additional sets in a timely manner.

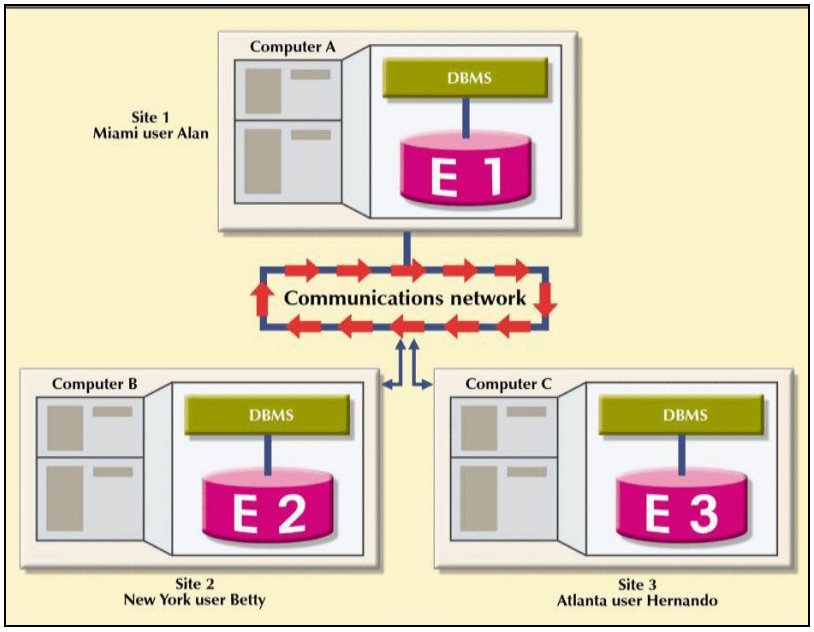

Figure 2. Distributed Database System

II. DEADLOCK FOR DISTRIBUTED SYSTEMS

A Distributed system consists of a collection of sites that are interconnected through a communication network each maintaining a local database system. The same conditions for deadlocks in uniprocessors apply to distributed systems. Unfortunately, as in many other aspects of distributed systems, they are harder to detect, avoid, and prevent. Distributed deadlocks can occur in distributed systems when distributed transactions or concurrency control is being used. A system is deadlocked if and only if there exists a directed cycle in the wait-for Graph (WFG)

Wait-For-Graph (WFG)

- Nodes – Processes in the system
- Directed Edges – Wait-For blocking relation
- A Cycle represents a Deadlock
- Starvation - A process execution is permanently halted.

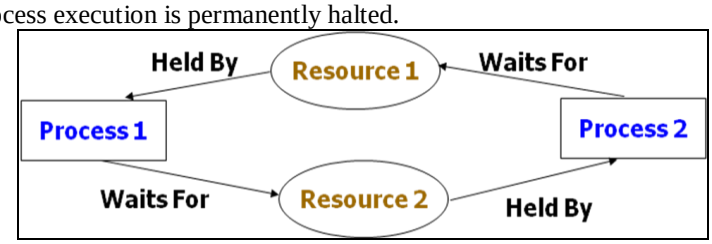

Figure 3 Wait for Graph

III. STRATEGIES FOR DEALING WITH DEADLOCKS

There are four strategies for dealing with distributed deadlocks:

a. **Ignorance**: ignore the problem (this is the most common approach).
b. **Detection**: let deadlocks occur, detect them, and then deal with them.
c. **Prevention**: make deadlocks impossible.
d. **Avoidance**: choose resource allocation carefully so that deadlocks will not occur.

IV. APPROACHES FOR DEADLOCK DETECTION IN DISTRIBUTED SYSTEMS

Deadlock detection[4] in distributed systems seems to be the best approach to handle deadlocks in distributed systems. The basic algorithm for distributed deadlock detection is as follows:

a. ***Obermarck's Path Pushing Algorithm:*** The basic idea underlying this class of algorithms is to build some simplified form of global WFG at each site. For this purpose each site sends its local WFG to a number of neighboring sites every time a deadlock computation is performed. After the local data structure of each site is updated, this updated WFG is then passed along, and the procedure is repeated until some site has sufficiently complete picture of the global situation to announce deadlock or to establish that no deadlocks are present. The main features of this scheme, namely, to send around paths of the

Individual sites maintain local WFGs

- Nodes for local processes
- Node "Pex" represents external processes that we don't know anything about

Deadlock detection

- If a site Si finds a cycle that does not involve Pex, it has found a deadlock.
- If a site Si finds a cycle that does involve Pex, there is the possibility of a deadlock.
  i. It sends a message containing its detected cycle to any sites involved in Pex
  ii. If site Sj receives such a message, it updates its local WFG graph, and searches it for a cycle
    a. If Sj finds a cycle that does not involve its Pex, it has found deadlock
    b. If Sj finds a cycle that does involve its Pex, it sends out a message **"Can report false deadlock".**

Figure 4, explains the working of this algorithm with help of example.

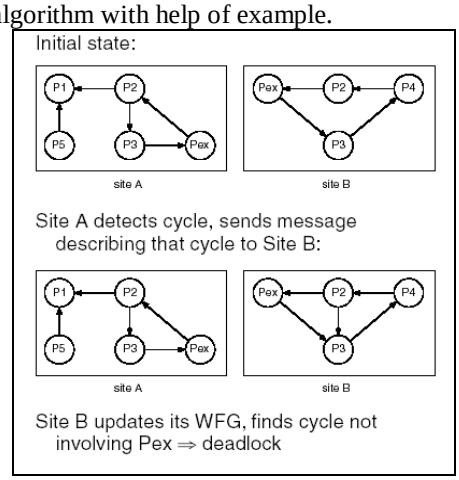

Figure 4. Detection of Cycle

***b. Chandy-Misra-Haas Edge-chasing algorithms*:** The presence of a cycle in a distributed graph structure can be verified by propagating special messages called probes along the edges of the graph. Probes are assumed to be distinct from resource request and grant messages. When the initiator of such a probe computation receives a matching probe, it knows that it is in cycle in the graph. A nice feature of this approach is that executing processes can simply discard any probes they receive. Blocked processes propagate the probe along their outgoing edges.

When a process has to wait for a resource (blocks), it sends a probe message to process holding the resource

- Process can request (and can have to wait for) multiple resources at once
- Probe message contains 3 values:
  a. ID of process that blocked
  b. ID of process sending message
  c. ID of process message was sent to

When a blocked process receives a probe, it propagates the probe to the process(es) holding resources that it has requested

- ID of blocked process stays the same, other two values updated as appropriate.
- If the blocked process receives its own probe, there is a deadlock.

Figure 5, explains the working of this algorithm with help of example.

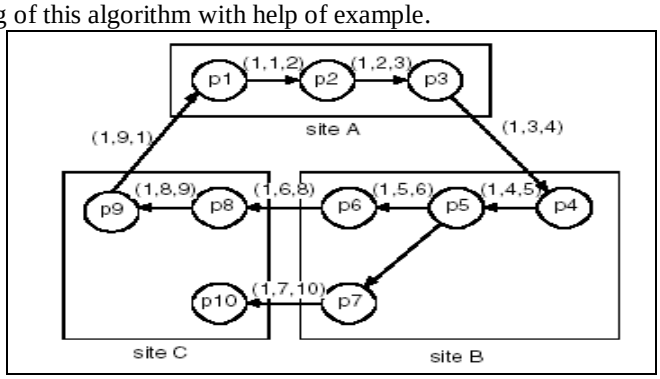

Figure 5. p1 initiates deadlock detection by sending a probe

V. CONCLUSIONS

In computer science, deadlock refers to a specific condition when two or more processes are each waiting for the other to release a resource, or more than two processes are waiting for resources in a circular chain. Deadlock is a

common problem in multiprocessing where many processes share a specific type of mutually exclusive resource known as a software lock or soft lock.

The problem of deadlock detection in distributed systems has undergone extensive study. In this paper we have tried to get rid of on distributed deadlock by studying the performance representative algorithms. We discuss distributed deadlock, deadlock dictation and prevention. In this way, the communication overhead of the deadlock detection procedure is reducing.

REFERENCES

[1] Chandy K.M. and Misra J. A distributed algorithm for detecting resource deadlocks in distributed systems. In Proc., A CM SIGA CT-SIGOPS Syrup. Principles of Distributed Computing, ACM, New York, 157-164 **(1982) .**

[2] Tamer M. Ozsu and Patrick Valduriez, Principles of Distributed Database Systems, Second Edition, Prentice-Hall, **(1999) .**

[3] Bernstein P. and Goodman N., Concurrency Control in Distributed Database Systems, ACM Computing Surveys 13/2 **(1981) .**

[4] Gligor V.D. and Shattuck S.H., Deadlock detection in distributed systems, IEEE Trans. Softw., Eng. SE-6, 5 435-440 **(1980) .**

