Research Journal of Recent Sciences ISSN 2277-2502
Vol. **1 (ISC-2011),** 422-425 **(2012)** *Res.J.Recent Sci.*

Mini Review Paper

Approaches for Deadlock Detection and Deadlock Prevention for Distributed systems

Gupta Dhiraj and Gupta V.K.
Department of Computer Science, NIMS University, Jaipur, Rajasthan, INDIA

Available online at: www.isca.in
(Received 11th October 2011, revised 27th February 2012, accepted 30th March 2012)

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: Distributed real-time databases, mobile real-time databases, concurrency control, data similarity, transaction scheduling.

Introduction

In the non-distributed case, all the information on resource usage lives on one system and the graph may be constructed on that system. In the distributed[1] case, the individual subgraphs have to be propagated to a central coordinator. A message can be sent each time an arc is added or deleted. If optimization is needed, a list of added or deleted arcs can be sent periodically to reduce the overall number of messages sent.

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 is Deadlock: A deadlock[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.

In above figure, Deadlock condition is shown, there are two processes P1 and P2 and two resources R1 and R2. Resource R1 is assign by process P1, held by process P2 and R2 is assign by process P2, held by process P1. Deadlock is present when the graph has cycles

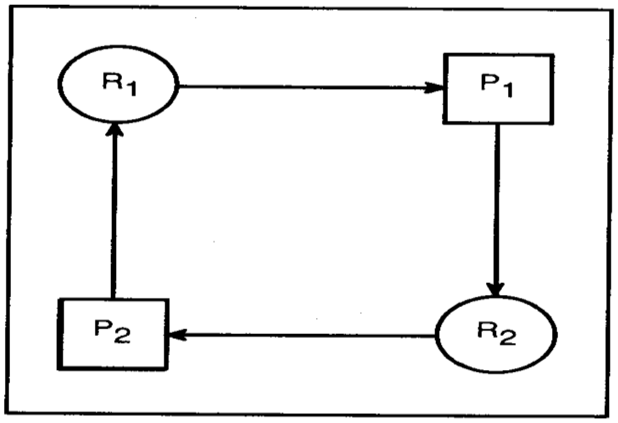

Figure-1
Deadlock

Distributed Database system: A distributed database management system ('DDBMS') is a software system that permits the management of a distributed database and makes the distribution transparent to the users.

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.

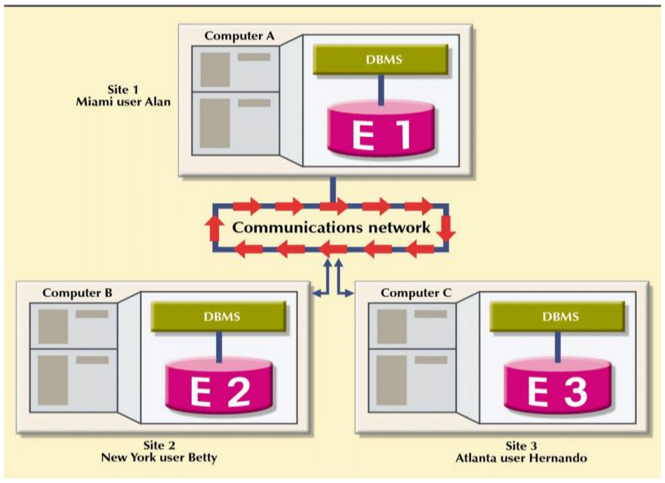

Figure-2
Distributed Database Environment

Deadlock for Distributed System: 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). If we have 3 computers 1, 2, and 3, with resources respectively, A, B, and C and D and we have three transactions T1, T2, and T3 that execute as indicate below:

In the above WFD that has a directed cycle, thus we have a distributed deadlock. A deadlock is a fundamental problem in distributed systems. A process may request resources in any order, which may not be known a priori and a process can request resource while holding others. If the sequence of the allocations of resources to the processes is not controlled, deadlocks can occur.

Strategies for dealing with distributed deadlocks: Distributed deadlocks can occur in distributed systems when distributed transactions or concurrency control is being used. There are four strategies for dealing with distributed deadlocks:

Ignorance: ignore the problem (this is the most common approach).

Detection: let deadlocks occur, detect them, and then deal with them.

Prevention: make deadlocks impossible.

Avoidance: choose resource allocation carefully so that deadlocks will not occur.

Table-1
Deadlock Process

T T1 (Transfer from B to A and D)	**T T2 (Transfer from C to B)**	**T T3 (Transfer from A to C)**
Lock (D), Deposit to D		
	Lock (B), Deposit to B	
Lock (A), Deposit to A		
		Lock (C) Deposit to C
Lock(B)		
	Lock (C)	
		Lock(A)
Deadlock		
Withdraw from B Unlock A,B,D	Withdraw from C Unlock B,C	Withdraw from A Unlock A,C

We have the wait-for graph (WFG) of distributed deadlock.

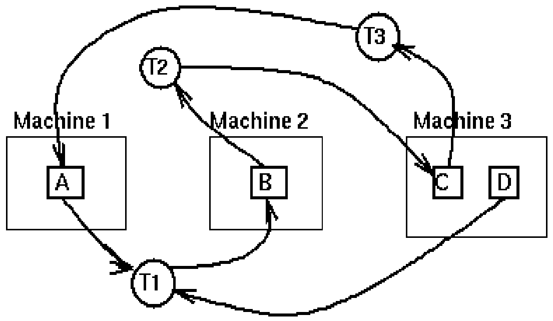

Figure-3
WFG

Approaches for deadlock detection for distributed systems: Deadlock detection[4] requires examination of the status of process-resource interactions for presence of cyclic wait. Deadlock detection 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:

Create the local wait for graph (WFG), Add possible edges obtained from other sites that may cause deadlock, The local WFG now also contains locks on remote objects and the sub transaction holding those lock, Determine the cycles, if it is to be found. There is a deadlock.

Path-pushing algorithms: 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 global WFG, have led to the term *path-pushing algorithms.*

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.

Distributed deadlock prevention: Deadlock prevention[4] protocols ensure that the system will never enter into a deadlock state. The basic prevention strategies are:

The strategies require that each transaction lock its entire data item before it begins execution. They impose partial ordering of all data item and require that a transaction can lock data item only in the order specified by the partial order. An alternative to detecting deadlocks is to design a system so that deadlock is impossible. One way of accomplishing this is to obtain a global timestamp for every transaction (so that no two transactions get the same timestamp). When one process is about to block waiting for a resource that another process is using, check which of the two processes has a younger timestamp and give priority to the older process.

If a younger process is using the resource, then the older process (that wants the resource) waits. If an older process is holding the resource, the younger process (that wants the resource) kills itself. This forces the resource utilization graph to be directed from older to younger processes, making cycles impossible. This algorithm is known as the wait-die algorithm. An alternative method by which resource request cycles may be avoided is to have an old process preempt (kill) the younger process that holds a resource. If a younger process wants a resource that an older one is using, then it waits until the old process is done. In this case, the graph flows from young to old and cycles are again impossible. This variant is called the wound-wait algorithm.

Conclusion

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)**

5. Dijkstra N.W, and Scholten C.S., Termination detection for diffusing computations, *Inf. Process. Lett.,* **11(1)**, 1-4 **(1980)**

6. Goldman B., Deadlock detection in computer networks. Tech. Rep. MIT-LCS-TR185, Massachusetts Institute of Technology, Cambridge, Mass., **(1977)**

7. Gray J.N., Notes on database operating systems. In Operating Systems: An Advanced Course, Lecture Notes in Computer Science, Springer-Verlag, New York, **60**, 393-481 **(1978)**

8. Science Dept., Univ. of Texas at Austin, July (1981). 7. HOARE, C.A.R. Communicating sequential processes. *Commun. ACM21,* **8**, 666-677 **(1978)**

9. Marsland T.A., and Isloor S.S. Detection of deadlocks in distributed database systems **(1980)**

10. Chandy K.M. and Misra J., A Distributed Algorithm for detecting Deadlocks in Distributed Systems, 157-164 **(1982)**

11. Elmasri Navathe, Database Concepts By Pearson Education, **(2011)**

12. Colloly., Data base Concepts By Pearson Education **(2011)**

13. Coronel Rob, Introduction to Database Concepts **(2011)**