www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 2 Issue 10 October,2013 Page No. 2900-2902

Deadlock Detection and Removalin Distributed Systems

Nidhi Sharma, Amol Parikh

Vellore Institute of Technology, Vellore, TamilNadu– 632014
Email: nidsharma1993@gmail.com
Vellore Institute of Technology Vellore, TamilNadu- 632014
Email: amol.chunky@gmail.com

Abstract

Advancement in information technology has been tremendous in the recent years. Distributed computing concepts are being applied to the database systems as well, leading to the development of distributed databases.[5] A distributed database is a collection of multiple logically interrelated databases distributed over a computer network. With the making of distributed databases rose the problem of deadlocks, concurrency and management of databases. In an operating system, a ***deadlock*** *is a situation which occurs when a process enters a waiting state because a resource requested by it is being held by another waiting process, which in turn is waiting for another resource.[4] 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.****Concurrency*** *refers to the process of simultaneous accesses to the database by various processes. One needs to synchronize the accesses to ensure consistency and avoid deadlocks.[4] In this paper, we discuss the various deadlock detection techniques and explain a new approach developed to detect and solve deadlocks.*

KEYWORDS

TWFG – Task Wait-For Graph

TRG – Task Resource Graph

RAG – Resource Allocation Graph

Deadlock Detection , Concurrency Control

1. INTRODUCTION

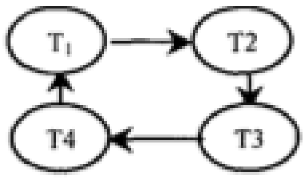
The advantages of distributed databases are sharing data among various users, restricting unauthorized access, providing storage structures for efficient query processing, providing backup and recovery services, providing multiple interfaces to different classes of users, representing complex relations among data etc. To fully utilize these benefits of the database systems, we need to deal with the problems of concurrency and deadlock.

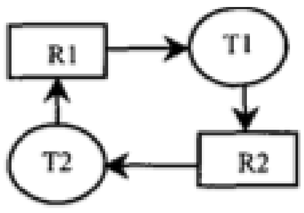
For a deadlock to occur there are four necessary conditions[1][4]:

(i) No Pre-emption
(ii) Mutual exclusion (resource can be accessed by only one process at a time)
(iii) Blocked waiting (a process gets blocked while waiting for a resource)
(iv) Hold and Wait (a process already holds some resources while requesting new ones)

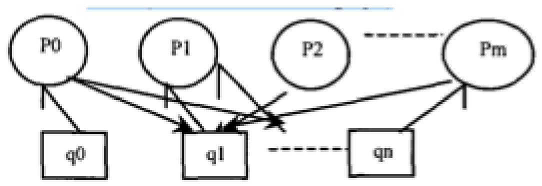
Concurrency issues are solved using lock-based algorithms while deadlocks are solved using graphs. Two main graphs used here for deadlock detection are Wait-for and Resource allocation.[1]

Wait-for graph: It is a mathematical tool used to express deadlocks.[1][2] A systems state can be represented using Task Wait-for graphs (TWFG). It is a directed simple graph in which an edge directed from a task t1 to t2 represents that t1 is waiting for t2. Another type of wait-for graph is Task resource graph (TRG).[1] There are two types of nodes – resources and processes. And there are two types of edges – request edge (directed from a process to a resource when the process requests a certain resource) and grant (directed from resource to the process representing that the process is granted access to that resource).[1]





Resource allocation graph: Deadlock detection can be represented by a RAG, which has two set of nodes namely processor nodes and resource nodes. A node is a vertex and the edges are used to show relations between various nodes. A node is called a sink when all its edges are incoming, and it is called a source if all its edges are outgoing. A path represents a sequence of nodes. If a path starts and ends at the same node then a deadlock is detected. [1]



2. ALGORITHMS FOR DEADLOCK DETECTION

The software solutions to the problems of deadlocks have been implemented using different types of algorithms, namely centralized, distributed and hierarchical. [1]

Centralized algorithms are implemented on the control site that has the responsibility of global system state information and searching for deadlocks whereas in distributed deadlock detection all nodes cooperate to detect deadlocks. [1] In hierarchical algorithms, the sites are organized in to hierarchy where each site is responsible for detecting deadlocks for itself and its children.[1]

Some of the distributed deadlock detection algorithms are discussed here.

(i).Path-Pushing algorithm:Basic idea is to construct a simplified form of system state graph at each site that is sufficient to detect a deadlock.[1] Here, after construction of these system graphs at each site, they are sent to the neighbors where the graphs from others and their own graphs are combined and sent to the next neighbor. This way finally a node has enough information to detect if a deadlock has occurred.

(ii).Edge-Chasing algorithm: In this algorithm, a node generates a probe message which is propagated along the system state graph. If the probe is finally received by its initiator then a cycle is detected.[1] If a node is not waiting for any resource it simply discards the probe message, others propagate it to the node whose resource they are waiting for.

3. BRIEF EXPLANATIONS AND COMPARISIONS OF SOME DEADLOCK DETECTION ALGORITHMS:

3.1 Deadlock Detection and Solving Algorithm

A directed graph can be represented with an adjacency matrix. Herethe adjacency matrix is modified a bit. The first column represents all processes while first row represents all the resources. The table is then filled according to RAG of the given system. The r represents a request made by process P for some resource (whose column is marked with ‘r’) R, and g represents a grant edge, i.e. the resource is granted to the process.

Now we define a few terms that we would need to further explain this algorithm.

Source is a process with n number of requests but no grants or with a grant but no requests. **Sink** is a process with one or more grants but no requests or one or more requests but no grants**. Link** is a node that has exactly one incoming edge and one outgoing edge. **Branch** is a node with one or more incoming and outgoing edges. A **cycle** is an ordered sequence of vertices such that except the first and the last ones no others are the same.[5]

Since the aim of the algorithm being discussed is detection and also solving deadlocks certain steps need to be followed to achieve the goal.

(i)Firstly a Resource Allocation Graph is constructed which shows all the resources allocated to a process and asked for by a process.

(ii) Next an algorithm is used to reduce this matrix as much as possible. Run a loop until all the vertices up to the terminal vertices are checked. Inside the loop, find the next state of the process by applying the formula to it. Using this formula the rows and columns containing all zeros, source and sink nodes are removed.[1] The state obtained from this computation is made the current state and the same computation is continued for all the states.

(iii)If there are no rows and columns left in the graph it means that no deadlock was detected. But if a row remains we detect at least one deadlock in our graph.[1]

(iv)This detected deadlock is now solved by using another algorithm. This algorithm also works in the same way as the one mentioned in step (ii) , except for further reducing the graph here, start removing the edges between the nodes(links and branches) until we reach the last row. This algorithm solves deadlocks by making processes to give up resources granted to them and also take away the requests that these processes have made. This is done until the deadlock detected is removed.

3.2 Two Phase Deadlock Detection Algorithm

In the original two-phase detection algorithm, every site maintains a status table which records the requests and grants that its processes have received (i.e. the transactions made by each process are recorded in this table).[2] A control site periodically collects these status tables from all the sites by broadcasting a message. It then creates a Wait-for graph for the system. If it contains a cycle it again takes the tables of all sites and considers only those nodes that appeared in the cycle in first phase. If they again form the same cycle, a deadlock is detected.[2]However, false deadlocks can also be detected using this method. Hence a modified version of this algorithm was made.

In this version each transaction’s edge has three values to describe it- T, R, t, where T represents the transaction, R represents the resource requested for or granted and t represents the local clock value of the transaction. The rest of the algorithm goes the same way as the original two-phase algorithm. This algorithm ensures that no false deadlocks are detected.[2]

3.3 One Phase Algorithm:

The control site here requests the tables only once, but two types of tables have to be transferred, namely transaction status table (of the processes) and the resource status table (maintained by the resource manager for each resource).[2] These tables are used to construct a global system state which takes into consideration only those nodes that appear in both the tables. Hence false deadlocks are not detected here. After creating the global system state if any cycles are found a deadlock is detected.[2]

In the modification of this one phase algorithm, each process maintains three records in a table, R, s, T_block, where R is the resource held /waited upon by the process, s is the status of the process which is ‘a’ if the resource is assigned to it or ‘w’ if it is waiting for the resource. T_block holds the value of time stamp at which the process requested the resource if it is blocked.[2]

When a process requests a resource, the resource manager checks if the resource is available, if yes, then the resource is assigned to the process through the time stamp message to the process and a lock is applied to the resource. Similarly when a process releases its resource then the resourced is unlocked and further assigned to another process if it was asked for; also the local time stamp of the resource manager is updated.[2] The control site broadcasts its request to all processes to send their time stamps, if the process is blocked it sends its T_block along with the process table to the control site. Here it is checked if the T_block received holds a value higher than the previous known local time stamp of the node by the control site.[2] If yes, then the edge P->R (process P waiting for resource R) is added to the WFG being made. If a cycle is formed then a deadlock is detected.[2]

The message complexity in this algorithm is halved from 2n in original one phase to n. Also the deadlock can be detected faster in this modified version as only process sends its graph to the control site.

3.4 A Token Based Algorithm for Deadlock Detection

This algorithm has a control process called the controller. This process is associated with all processes. The job of this controller is to coordinate with other controllers and help in the detection of deadlocks in the processes.[3]

Each process P has an input buffer where all the messages addressed to it are stored. This buffer is readable by the controller also. Here, a virtual ring is established. A value token is generated by the controller and passed on to the process next to it in the virtual ring.[3] This token carries the names of set of processes as PDs; these are the processes that the controller, according to its current knowledge thinks are deadlocked. A process is removed from the PD list only if it receives the token from the previous node. The token moves around the ring. If the value for the PD does not change when the token is monitored by the controller, it detects that a deadlock exists. But if the set PD becomes empty it concludes that no deadlock was there at the time t when it initiated the token. A Boolean variable 'cp' is maintained here, this variable if false indicates that a process P is active and if true means that the process is waiting. The controller can check and detect for sure if a process is active or not and accordingly set the value of 'cp' for this process. A deadlock is detected only if 'cp' is true and also the given process remained continuously passive since the last token visit.

4. CONCLUSION

The paper discusses some hardware and software algorithms for deadlock detection. These algorithms have some advantages like detecting a true deadlock, reducing the time for deadlock detection, implementing algorithms that solve these deadlocks etc, at the same time they face certain disadvantages too. One needs to carefully select an algorithm for implementation of deadlock detection in the distributed systems so that properly functional and efficient distributed databases can be built.

5.ACKNOWLEDGEMENTS

We would like to thank Prof Saritha. V for her expert advice in the area of Database Systems and Deadlock Detection and Concurrency Control

6. REFERENCES

[1]H. A. Ali**,** T. EL-DNAF, and M SALAH,*A Proposed Algorithm for Solving Deadlock Detection in Distributed Database Systems,* IEEE© 2004
[2]Ajay D Kshemkalyani, MukeshSinghal, *Correct Two-Phase and One-Phase Deadlock Detection Algorithms for Distributed Systems*Parallel and Distributed Processing, 1990. Proceedings of the Second IEEE Symposium
[3].JersiBrezezinski, Jean-Michel Helary, Michel Raynal, *Deadlock Models and General Algorithms for Distributed Deadlock Detection*. Journal of Parallel and DistributedComputingVolume 31, Issue 2, December 1995

[4].Silberschatz, Abraham (2006*). Operating System Principles (7 ed.).* Wiley-India. p. 237. Retrieved 29 January 2012.

[5] R. Elmasri, S.B. Navathe, *"Fundamentals of Database Systems",* Third Edition, 2000