© 2014 IJIRT | Volume 1 Issue 5 | ISSN : 2349-6002

DEADLOCKS IN DISTRIBUTED SYSTEMS

Yogesh Bhatia, Sanjeev Verma
Department of Information Technology
Dronacharya College of Engineering, Gurgaon, Hr

***Abstract*- In a distributed database environment, where the data is spread across several sites there are many concerns to deal with such as concurrency control, deadlock. Deadlock is defined as the permanent blocking of a set of processes that compete for system resources, including database records and communication lines i.e. it 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 other words processes are blocked, waiting on events that could never happen. Deadlocks impact the overall performance of the system. This paper describes the detection of deadlocks in distributed data bases-a hierarchically organized one and a distributed one. Deadlock detection and resolution is one among the major challenges faced by a Distributed System**

I. INTRODUCTION

A deadlock 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. Distributed database systems (DDBS) consist of different number of sites which are interconnected by a communication network. In such a resource sharing environment the database activities can be performed both at the local and global level so if the allocation of the resource is not properly controlled than it may lead to a situation that is referred to as deadlock. In Distributed database system model, the database is considered to be distributed over several interconnected computer systems. Users interact with the database via transactions. A transaction is a sequence of activities such as read, write, lock, or unlock operations.

Conditions for deadlock in resource allocation

– Mutual exclusion: The resource can be used by only one process at a time
– Hold and wait: A process holds a resource while waiting for other resources
– No preemption: A process cannot be preempted to free up the resource
– Circular wait: A closed cycle of processes is formed, where each process holds one or more resources needed by the next process in the cycle

II. THE FOUR STRATEGIES FOR HANDLING DEADLOCKS

- The Ostrich algorithm
  - No dealing with the problem at all is as good and as popular in distributed systems as it is in single-processor systems.
  - In distributed systems used for programming, office automation, process control, no system-wide deadlock mechanism is present -- distributed databases will implement their own if they need one.
- Deadlock detection and recovery is popular because prevention and avoidance are so difficult to implement.
- Deadlock prevention is possible because of the presence of atomic transactions. We will have two algorithms for this.
- Deadlock avoidance is never used in distributed system, in fact, it is not even used in single processor systems.
  - The problem is that the banker's algorithm need to know (in advance) how much of each resource every process will eventually need. This information is rarely, if ever, available.

III. DEADLOCK AVOIDANCE

Deadlock avoidance is an approach in which deadlocks are dealt before they occur. All the possible ways are taken to avoid the deadlock. When a transaction requests a lock on the data item that has already been locked by some another transaction in an incompatible mode, the deadlock avoidance algorithm decides if the requesting transaction can wait or if one of the waiting transactions need to be aborted.

IV. DEADLOCK PREVENTION

It is an approach that prevents the system from committing an allocation of locks that will eventually lead to a deadlock. This technique requires pre- acquisition of all locks. The transactions are required to lock the entire data item that they need before execution. Deadlock prevention deals with deadlock ahead of time. Thus it may prevent a deadlock to occur.

V. DISTRIBUTED DEADLOCK DETECTION

In this approach, deadlock may have already occurred and the deadlock detection technique tries to detect it and gives the process by which it can be resolved. Thus the system periodically checks for them. The existence of a directed cycle in the Wait-for-Graph indicates a deadlock. To break the deadlock cycle the victim transaction is selected, which is then aborted to make the system deadlock free.

- Since preventing and avoiding deadlocks to happen is difficult, researchers works on detecting the occurrence of deadlocks in distributed system.
- The presence of atomic transaction in some distributed systems makes a major conceptual difference.
  - When a deadlock is detected in a conventional system, we kill one or more processes to break the deadlock --- one or more unhappy users.
  - When deadlock is detected in a system based on atomic transaction, it is resolved by aborting one or more transactions.
  - But transactions have been designed to withstand being aborted.
  - When a transaction is aborted, the system is first restored to the state it had before the transaction began, at which point the transaction can start again.
  - Thus the difference is that the consequences of killing off a process are much less severe when transactions are used.

VI. ALGORITHMS

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.

VII. DEADLOCK RECOVERY

Once the deadlock algorithm has successfully detected a deadlock, some strategy is needed for recovery. There are various ways:

1. Recovery through Preemption : In some cases it may be possible to temporarily take a resource away from its current owner and give it to another.
2. Recovery through Rollback: If it is known that deadlocks are likely, one can arrange to have processes checkpointed periodically.
3. Recovery through Termination: The most trivial way to break a deadlock is to kill one or more processes. One possibility is to kill a process in the cycle

REFERENCES

1. Approaches for Deadlock Detection and Deadlock Prevention for
Distributed systems, Gupta Dhiraj and Gupta V.K. Research Journal of Recent Sciences. ISSN 2277-2502

2. Deadlock Detection Techniques in Distributed Database
System, Swati Gupta. International Journal of Computer Applications (0975 – 8887) Volume 74– No. 21, July 2013.

3. Locking and Deadlock Detection in Distributed Data Bases, DANIEL A. MENASCE AND

RICHARD R. MUNTZ. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-5, NO. 3, MAY 1979

4. Analysis for Deadlock Detection and Resolution Techniques in
Distributed Database, Swati Gupta , Meenu Vijarania. International Journal of Advanced Research in Computer Science and Software Engineering . Volume
3, Issue 7, July 2013 ISSN: 2277 128X

5. Distributed Operating Systems by SHUBHRA GARG.