A NEW APPROACH FOR DEADLOCK DETECTION AND REMOVAAL FROM DISTRIBUTED DATABASE SYSTEMS

Mohit Bhardwaj[1], Dr. Parul Tomar[2]

[1,2]YMCA University of Science and Technology, Faridabad, Haryana (India)

ABSTRACT

A Distributed database system (DDBS) is a collection of databases distributed over several sites interconnected by a communication network. It provides a resource-sharing environment where database activities can be performed optimally both in global and local framework. The distributed nature of database demand full proof control structure for its proper and effective functioning. Therefore the allocation of the resources should be properly controlled otherwise it may lead to several anomalies such as concurrency of transaction, synchronizing of events and deadlocks. A deadlock is a state where some processes request for some resources but those resources are held by some other processes. Occurrence of deadlock in a system will lead to resource wastage and breakdown of the system. This paper provides a new approach for detection and removal of deadlock in distributed databases.

Keywords: Database, Distributed Database, Deadlock

I. INTRODUCTION

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 [1]. In Distributed database system model, the database is considered to be distributed over several interconnected computer systems. Users interact with the database via transactions[2,3].Execution of transaction can lead to deadlocks in the system.

A deadlock is a situation in which the processes holding some resources request for the access of resources held by other processes in the same set. The simplest illustration of a deadlock consists of two processes, each holding a different resource in exclusive mode and each requesting an access to resources held by other processes. Unless the deadlock is resolved, all the processes involved are blocked indefinitely. Therefore, a deadlock requires the attention of a process outside those involved in the deadlock for its detection andresolution.

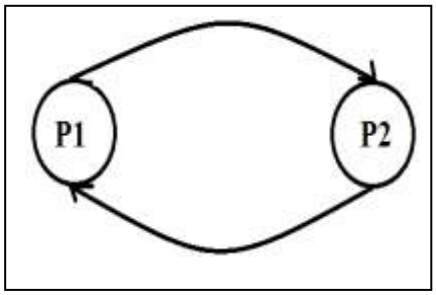

Figure 1 Deadlock Example

In Figure 1 a simple deadlock condition is shown. Assuming two processes P1 and P2 requested some resources for their computation. If the resource requested by process P2 is held by process P1 then P2 would be waiting for P1 to release the required process. Similarly at the same time if the resource requested by Process P1 is held by P2 and P1 is waiting for P2 to release the requested process. This situation leads to a deadlock.

Distributed database systems are very prone to deadlocks as different sites are unable to keep track of the transactions running at other sites[4].

There are three different techniques to handle the deadlocks[5]:

- Prevention: This technique prevents the system from making any deadlock. In this technique information of the all resources which are allocated to some process is recorded. Now if a process requests for some resources, the system will grant only when all the resources are available. System will make sure that not a single resource which is requested is required or hold by some other process.
- Detection: This technique is used to detect existing deadlock in the system. When Resource allocation is fair and processes holding and waiting for resources results in deadlock. When a deadlock occurs it should be detected and resolved as soon as possible for good efficiency of the system.
- Removal: When a deadlock is detected in a system, it must be removed by terminating some process. Removal of deadlock needs the roll back facility to the terminated process. There are various strategies for deadlock removal such as time stamping, youngest process removal, priority based removal etc.

II. ISSUES IN DEADLOCK DETECTION

- Deadlock detection involves two basic tasks: maintenance of the state graph and search of the state graph for the presence of cycles. Categorization of deadlock detection algorithm largely depends upon the manner in which the state graph information is maintained. There are three types of algorithms for deadlock detection in distributed systems[6]:
- Centralized algorithms: In centralized algorithms the state graph is maintained at a single designated site, which has the sole responsibility of updating it and searching it for cycles.
- Distributed algorithms: In distributed algorithms the state graph is distributed over many sites of the system, and a cycle may span state graphs located at several sites, making distributed processing necessary to detect it.
- Hierarchical algorithms: In hierarchical algorithms sites are arranged in a hierarchy, and a site detects deadlocks involving only its descendant sites. Hierarchical algorithms exploit access patterns local to a cluster of sites to efficiently detect deadlocks.
- In a distributed system it is very difficult to identify the deadlocks as there is no global memory and communication occurs solely by messages. It is difficult to design a correct deadlock detection algorithm because sites may receive out-of-date and inconsistent state graphs information of the system
- The next section will give a new algorithm which will check for deadlock in local system first and then will detect the deadlock in a distributed system.This protocol will remove the possibility of false deadlock and will detect all the deadlocks in the system.

III. PROPOSED MODEL

A process P arrives at site j and it request some resources. This request can be for any data from any site. Rp Denotes requested data (R1 R2 R3) where these sites R1 R2 R3 ... can be at any site i.e. process P could request some data from its home site and some data from some other site or it could request all data from home site or all data from other sites. After process P's arrival algorithm counts total number of resources requested by the process P and stores it in a variable "total".

```
A Process P arrives at site j and requests some resources.
Rp Denotes requested data from various (R1 R2 R3 …) sites by new process P.
Total = count (Rp);
```

For each requested data by new comer process P, algorithm checks whether this requested data belongs to the home site or not. If it belongs to the home site then localAllocation function will be called. Requested data is passed in localAllocation function. If the requested data belongs to some site other then the home site then globalAllocation function will be called.

```
For (i=0; i<total; i++) {
        If (xi belongs to Sj) {
                localAllocation(xi);
        }
        Else {
                Status = checkLocalDeadlock();
                If(Status == NoLocalDeadlock) {
                        globalAllocation(xi);
                }
                Else {
                        Print("Can't Allocate as local deadlock is present");
                }
        }
}
```

If the requested resource belongs to the home site then algorithm will check whether it is free or not. If it is free then algorithm will call checkLocalDeadlock function to check whether there is some local deadlock or not and then system will allocate Resource to Process P. If Resource requested at home site by Process P is not free then algorithm first call checkLocalDeadlock function and change the status that Process P is waiting for Resource.

```
        Function localAllocation(Ri) {
        If (Ri is free) {
                checkLocalDeadlock();
                Allocate Ri to P;
        }
        Else {
                checkLocalDeadlock();
                P waiting for Ri;
        }
        }
```

If the requested resource belongs to some site other then the home site then algorithm will check whether the requested resource is free or not. If the requested resource is free then the algorithm will call checkGlobalDeadlock function to check whether there is some global deadlock or not and then system will

allocate Resource to Process P. If Resource requested at global site by Process P is not free then algorithm first call checkGlobalDeadlock function and change the status that Process P is waiting for resource Ri.

```
Function globalAllocation(Ri) {
        If (Ri is free) {
                checkGlobalDeadlock();
                Allocate Ri to P;
        } Else {
                checkGlobalDeadlock();
                P waiting for Ri;
        }
}
```

checkLocalDeadlock function will first create a local vector array. A local vector array is a series of processes waiting for one another. For example: P1 -> P2 -> P3 -> P4 -> P5, here P1 is waiting for P2, P2 for P3, P3 for P4 and P4 for P5. Now for each vector array formed, algorithm will check for circular wait in the vector array. If one process comes twice in a single vector array then it is called "localCircularWait" and it results into a deadlock. If any deadlock found removeDeadlock function is called by passing the value of that vector array to the function.

```
Function checkLocalDeadlock() {
        Create Local Vector arrays VAi;
        Foreach(VAi) {
                Starting from Process P
                If (localCircularWaitFound) {
                        Deadlock occurred;
                        removeDeadlock(VAi);
                }
}
}
```

checkGlobalDeadlock function will first create a global vector array. A global vector array is a series of processes waiting for one another at different sites. Now for each global vector array formed, algorithm will check for circular wait in the vector array. If one process comes twice in a single global vector array then it is called "globalCircularWait" and it results into a global deadlock. If any deadlock found removeDeadlock function is called by passing the value of that global vector array to the function.

Main difference in checkLocalDeadlock and checkGlobalDeadlock function is, checkLocalDeadlock make a local vector array for a single site on which it is called. All the transactions on that site are taken into account to check for a deadlock at that site. On other hand checkGlobalDeadlock function creates a global vector array which includes two or more sites. All the transactions of included sites are taken into account to check for a global deadlock in the system.

```
Function checkGlobalDeadlock() {
        Create Global Vector arrays VAi;
        Foreach(VAi) {
                Starting from Process P;
                If (globalCircularWaitFound) {
                        Deadlock occurred;
                        removeDeadlock(VAi);
                }
        }
}
```

If there is a deadlock in our system, whether a local or a global, it must be removed as soon as possible to make our system deadlock free and to make it run properly without any error.removeDealock function takes a vector array as an input. This vector array contains the list of processes involved in the deadlock and making the system stand still.

Starting from the first process, eliminate it from the list and check for the existence of deadlock in the rest of the processes. If there is no deadlock present then terminate the eliminated process and make our system deadlock free.

If there is still a deadlock in our system, First we will include last terminated process in our vector array again and then move to next process and eliminate that process and check for the deadlock in rest of the processes including the previous eliminated process. If no deadlock found terminate this process and make our system deadlock free if not then move the next process and repeat the above process till no deadlock in the system.

Process of removing a local or a global deadlock is same we have to check for a cycle in our vector array which is formed using DTS or LTS and a transaction which will give less over head will be removed.

3.1 Algorithm for Deadlock Removal is Given Below

```
Function removeDeadlock(VA) {
        Make a list of processes in Vector Array (VA).
        Foreach (process p) {
                Eliminate process p;
                Create a vector array for the rest of the processes.
                If one process occurred twice in the vector array
CircularWaitFound = true;
Else
CircularWaitFound = flase;
                If (CircularWaitFound) {
                        Include the last eliminated process;
                        Continue;
                }
                Else {
                        Terminate process P.
                        Deadlock Removed;
                }
        }

}

}
```

IV. CONCLUSION

Deadlocks in distributed databases are very hard to determine. Some times this determination can lead to the detection of false deadlocks and sometimes some deadlocks remain undetected. Using this newly proposed algorithm the number of messages sent over the network has been reduced by putting a condition at local site that if there is a deadlock at local site then request for global allocation won't be entertained. Also in order to remove deadlock newest transaction won't be removed to, a transaction which results in zero deadlocks in the system will be terminated.

REFERENCES

[1]. Wikipedia (http://en.wikipedia.org/wiki/Database).

[2]. Amjad Umar, "Distributed Database Management Systems Issues and Approaches", Technical Report No. 88-8, July 1988.

[3]. MasoomehGhodrati, , Ali Harounabadi, "Provide a New Mapping for Deadlock Detection and Resolution Modeling of Distributed Database to Colored Petri Net", International Journal of Computer Applications (0975 – 8887), Volume 95– No.5, June 2014.

[4]. B.M. MonjurulAlom, FransHenskens, Michael Hannaford, "Optimization of Detected Deadlock Views of Distributed Database", International Conference on Data Storage and Data Engineering,2010.

[5]. MukeshSinghal, Deadlock Detection in Distributed Systems, SURVEY & TUTORIAL SERIES, Nov. 1989

[6]. JersiBrezezinski, Jean-Michel Helary, Michel Raynal," Deadlock Models and General Algorithms for Distributed Deadlock Detection", Journal of Parallel and DistributedComputing, Volume 31, Issue 2, December 1995.