A Review Paper on Deadlock Detection in Distributed System

Palak Arora
Deptt. of Computer Science Engineering,
FET, MRIU, Faridabad
Accendere Knowledge Management Service Pvt. Ltd., India
E-mail: palakarora307@gmail.com

Abstract: If we don't have the distributed shared memory then the systems are easily prone to deadlocks. Deadlock is basically the result of uncontrolled sequence of request and release of resources among a number of different processes in distributed system. This survey paper presents some of the system models like and , or , unrestricted etc. and some selected deadlock detection algorithms like Chandy and Mishra , Obermarck etc. are also presented to see how in deadlocks are detected in different types of algorithms

Indexed Terms: Database, Distributed database, Deadlock, Deadlock detection

I. INTRODUCTION

Database is nothing but a set of related data and the way in which it is organised. It usually provides an integrated set of computer software which allows the users to interact with one or more databases and provides access to all the data contained in it. It is the most important part of the software industry as it provides various functions that allow us to enter the data/information, store it and retrieve it in large quantities and also allows us to manipulate it. It is used to store large amount of data such as engineering data, economic models etc.

There are two categories of database i.e non-distributed and distributed database. Non-distributed databases are defined as the database on a single site and can be accessed from there only but in case of distributed databases they are distributed over a number of sites which are interconnected with each other through communication network. It provides us a resource-sharing environment where database activities can be performed optimally. Each site is built up of a local database and its transactions running on them. Although the sites are dispersed, a distributed system manages and controls the entire database as a single collection of data. In distributed database, deadlocks is the biggest concern, as a single database is accessed from various sites and the number of persons demand the allocation of same resources at the same time which may lead to a deadlock condition. Deadlock detection is an important problem in database systems, and has gained a lot of attention in the research community. Normally, a deadlock situation is defined as the maximum possible result of competition for transaction and resources, for example: at a time a number of database transactions request the exclusive access to the same data items. There are several interesting components of deadlock problem. These are deadlock prevention, deadlock avoidance. In accordance with deadlock detection their main aim is to search a cycle or knot. After finding the cycle or knot there occurs the selection of a so-called victim whom we have to roll- back or abort in order to break the deadlock, and finally in this way the deadlock is able to resolute itself. Deadlock prevention means that a process should acquire all the resources at once by a transaction. The requirement of these resources can't be always fulfilled in a database environment as the resource needs transaction that may be data dependent and not precisely known at the starting. Therefore, it is a necessary condition for a transaction to acquire all possible resources required by it, hence it also reduces the system concurrency. Deadlock avoidance is defined as a technique which is used to avoid the deadlocks. For this we need some advance knowledge of the resources used in order to determine that there is a valid sequence of transactions which are already initiated are running to completion or not.

In simple words deadlock is basically a condition when a process requests for a resource and that resource is occupied by some other process then this condition is called as deadlock. A process may occur in two states: (i) running and (ii) blocked. In running state (also called active state), a process acquires all the needed resources before moving towards the execution whereas in the blocked state, a process waits to acquire some resources. Wait for Graphs (*WFG*) are generally used for depicting Deadlocks, *WFG* is a directed graph that shows how one transaction is waiting for another transaction for its completion. In distributed systems, there are two types of *WFG* i.e. local *WFG* and global *WFG*. A cycle in local *WFG* tells us that a deadlock has occurred. Even if there is no cycle in local *WFG* it does not imply that no deadlock has occurred. The Global Wait for Graph can be classified in two parts: (1) A global wait for graph is needed to be maintained for the information passed through transactions.(2) When simply messages are sent among transactions and there is no need to maintain global wait for graph. [6]

Deadlock handling is very much difficult in distributed systems as none of the site is having a correct knowledge of the current or recent state because every inter site communication includes a finite and unpredictable delay. Deadlock prevention can be easily done by acquiring all the resources needed before it begins executing or by preempting a process which is holding the needed resource. This is an inefficient and impractical approach in distributed database systems. The deadlock avoidance approach is defined as when a resource is granted to a process only and only if the resulting global system state is totally safe. However, due to some problems, deadlock avoidance is impossible in distributed systems. Deadlock detection always requires the examination of the status of resource interactions in order to check the presence of cyclic wait. Deadlock detection in distributed systems is said to be the best approach to handle deadlocks in distributed systems.

II. LITERATURE REVIEW

2.1 Deadlock Detection

Since it is seen that deadlock prevention and avoidance algorithms are unsuitable for the distributed systems under consideration, deadlock detection methods must be examined. Deadlock detection is basically a process of determining wether a deadlock really exists or not and also the processes and resources involved in it. The basic idea behind deadlock detection is to check allocation against resource availability for all possible allocation sequences to determine if it is in a deadlock state. Various researchers have introduced various algorithms which are used for detection of deadlocks. It checks the status of process-resource interactions for presence of cyclic wait for graph. A wait for graph is nothing but a directed graph which is used to determining deadlocks in operating systems as well as databases. Detection of deadlocks involves addressing two issues.(1) Maintenance of the WFG. (2) Searching of the WFG for the presence of cycles. [6] In 1989 Singhal [16] explained that the correctness of an algorithm is measured by using 2 parameters: (1) Progress: The algorithm must detect all existing deadlocks in finite time. In other words we can say that there is no undetected deadlock. (2) Safety: There should not be any false deadlock which is detected. As we proceed for detection we need to classify them according to knapp.

2.2 Knapp's Classification[11]

KNAPP in 1987 classified the deadlock detection algorithms into 4 groups: path-pushing, edge-chasing, diffusion computation, global state detection. We have several deadlock detection algorithms such as HO's and Ramamoorthy algorithm stated in 1982, Obermarck algorithm also provided to us in 1982 and many more which comes under the above classification:-

Path-pushing algorithms: In this algorithm the detection of deadlocks is done by maintaining an global WFG explicitly. Building of global WFG for each site of the system represents the basic idea of this algorithm. In this, the system sends the local WFG to all its neighbouring sites when deadlock computation is performed at each site. After updating each site with its local data structure, then we need to pass the updated WFG to other sites and repeat the same process until we get a complete picture of global state and we are ready to announce a deadlock or declare that the site is deadlock free. This feature of sending around the paths of global WFG has led to the term path-pushing algorithms.

Edge Algorithms: In an edge-chasing algorithm, some special message called probes are propagated to check the presence of a cycle, along the edges of the graph. These special messages are different than the request and reply messages. A site can delete the cycle formation if and only if the previous matching probe is received by it. Whenever a probe message is received by the executing process, the message is discarded and the process continues. The probe messages are propagated by the blocked processes only along their outgoing edges. Advantage of this algorithm is that the messages are of very short size.

Diffusing Computations Based Algorithms: In this algorithm the computation is diffused through the WFG of the system to detect the deadlock. Here the detection of deadlocks is done by using echo algorithms. On the underlying distributed computation the above computation is super imposed. Here the deadlock is declared by the initiator after the termination of computation. Query messages are sent by the process along all the outgoing edges in the WFG. These queries are successively propagated (i.e., diffused) through the edges of the WFG.

Global state detection based algorithms: This algorithm uses the snap shot technique for detecting deadlock. It is based on the following facts: (1) Without freezing the underlying computation a consistent snapshot can be obtained. (2) Before the snapshot collection is initiated if there is a stable property hold in the system, then the property will still hold during the snapshot. Therefore, the snapshot technique was used in distributed systems to detect the deadlocks of the system and examining it for the deadlock condition's.

2.3 Models of Deadlocks [12]

Distributed systems allow several kinds of resource requests.

2.3.1 The Single Resource Model: In the single resource model, only and only one outstanding request for a unit resource is allowed.1 is the maximum possible out-degree of a node in a WFG for the single resource model, the cycle in the WFG shall indicate that deadlock is there.

Algorithm	Authors Name	(Year of Publication)
Mitchell and Merrit's	Mitchell and Merrit	1984

2.3.2 And Model: In this model, one or more requests can be made by the process simultaneously and the request is satisfied only after acquiring all the requested resources by the process. The out degree of a node in the WFG for AND model can be more than 1. The deadlock is indicated by the presence of a cycle in the WFG of AND model. Since in the single-resource model, a process can have only one outstanding request, the AND model is more commonly used than the single-resource model as it can have more than one outstanding requests.

Algorithm	Authors Name	(Year of Publication)
Obermarck	Obermarck	1982
Candy and Mishra	Candy and Mishra	1983

2.3.3 OR Model: In OR model, request can be made for numerous resources simultaneously by a process and the request is satisfied when even one of the numerous resources are acquired by a process. Presence of a cycle in the WFG of an OR model does not indicate that there is a deadlock in the OR model.

Algorithm	Authors Name	(Year of Publication)
Candy, Mishra and Hass	Candy, Mishra and Hass	1983

2.3.4 Unrestricted Model: here the underlying structure has no assumptions regarding the resource request. It is the most general model as it is assumed to be the most stable deadlock. This model helps us to separate the properties of the problem like stability and deadlock are separated from underlying distributed systems computations.

2.3.5 $\binom{P}{Q}$ Model: (called the P-out-of-Q model) provides us with a pool of n resources from which we have to choose k available resources. The expressive power is same as that of and-or model. However, ($^p{}_{q)}$ model gives us a very compact formation of a request. Every request in the ($^p{}_{q)}$ model can be expressed in the and-or model and vice-versa. Note: Requests for p resources can be stated as ($^p{}_{p)}$ in and Model and ($^p{}_{1)}$ in OR model.

Algorithm	Authors Name	(Year of Publication)
Bracha and Toueg's	Bracha and Toueg	1983
Singhal	Singhal	1994

III. DEADLOCK DETECTION ALGORITHMS

Some basic and important algorithms used for detecting deadlocks are as follows:

3.1 HO and Ramamoorthy Algorithm[4]

In 1982 HO and Ramamoorthy gave the HO's algorithm which maintains a status table for all the process that are initiated at the site. This algorithm has two phases. Here the table keeps the track of resources the process has locked with itself and the resources for which the process is waiting for. Here a site is chosen as a controller and the following two phases are created:

Phase1

Step 1: The controller broadcast a message requesting all the sites to send their status table.

Step 2: After receiving all the status table it constructs a wait for graph. If a cycle is detected then initiate the second phase else there is no deadlock and release its control .

Phase 2

Step 1: It is a verification phase it again broadcast a second message requesting everyone to send their status table.

Step 2: After receiving the entire message it constructs the wait for graph now if there is a cycle it reports a deadlock to the deadlock resolver or else the control is released.

3.2 Obermarck Algorithm[2]

In 1982 Obermarck gave an approach in which an external node t_{ext} is added to a local wait for graph to indicate the agent at the site. When a transaction $T1$ at site $S1$, creates a agent at site $s2$ then an edge is added from $T1$ to T_{ext} node to the local *WFG* at site $S1$ and from T_{ext} node to the agent of $T1$ at site $s2$. If the local *WFG* contains a cycle that does not include T_{ext} then the site is in deadlock. A global deadlock is detected if any local *WFG* contains a cycle including T_{ext} node. Then we have to merge the graphs in order to detect the deadlocks. If no cycle is found, then the process goes on until a cycle appears or entire global *WFG* is built and no cycle has been detected. The advantage of this algorithm is that the number of messages to be transmitted is less in comparison to HO's algorithm. Disadvantage: It can detect false deadlocks. Performance analysis: it requires η (η - 1) messages to be transmitted.

3.3 Chandy and Mishra ALGORITHM [10]

In 1982 Chandy and Mishra explained an algorithm which is based on edge-chasing. A special message called probe is used in this algorithm, which is a triplet *(i,j,k)* denoting that it belongs to a initiated deadlock detection of process P_i and it is initiated by sending the messages from home of process P_j to the home of process P_k. A global *WFG* graph is used by the message to travel through its edges, when a probe message returns to the process that initiated it then the deadlock is detected. A process P_j is dependent on another process P_k if there exists a sequence of processes P_j, P_i1, P_i2, ..., P_{im}, P_k, such that all the processes are blocked in the sequence except P_k in the sequence and each process, except the P_j, is holding the resources for which the previous process are waiting in the sequence. Process P_j is locally dependent upon P_k if P_j is dependent upon P_k and both the processes are present on the same site. The performance analysis of this algorithm is it requires η (η - 1) probes for N number of nodes. Its main drawback is that it can detect the false deadlock.

3.4 Chandy Mishra and Haas Algorithm [1]

In 1983 Chandy Mishra and Haas came up with a modified version of Chandy and Mishra algorithm. Previously a special message known as probe was used in a set of triplet *(i, j, k)* with the help of which we can detect the deadlock when the process comes back to the home i.e the point from where it is initiated i.e. P_i. They used the diffusion computation technique in the modified version to detect the deadlock in the blocked processes. Here the deadlock computation has two forms of messages (i) query *(i, j, k)* and (ii) reply *(i, j, k)*. Both these messages denote that they belong to a process intiated by P_i and the messages are transferred from process P_j to P_k. Here the deadlock detection is initiated by the blocked process by transferring the query messages in its dependent set to all the processes. When a query or reply message is received by the active process, it discards it. When P_k which is a blocked process receives a query *(i, j, k)* message, it takes the following actions: (1) If this is the first query message received by P_k for the deadlock detection initiated by (called the engaging query), then it propagates the query to all the processes in its dependent set and sets a local variable *numk (i)* to the number of query messages sent. (2) If the query is not engaging, then P_k returns a reply message to it immediately provided P_k has been continuously blocked since it received the corresponding engaging query. Otherwise, it discards the query.

3.5 Bracha and Toueg's Algorithm [8]

In 1983 Bracha and Toueg presented an algorithm in which the initiator initiates a snapshot to compute the wait-for graph. Each node u takes a local snapshot of the requests which were sent or received that weren't yet granted or purged, grant and purge messages in edges. Then it computes: $Output_u$: the nodes it sent a request to (not granted) $Input_u$: the nodes it received a request from (not purged). The algorithm can be reffered as echo-like algorithm where requesters send out notifications, indicating their resource requirements and requests grant resources if they have any. At the end, if there are ungranted requests, it implies that the basic algorithm has deadlocked.

Algorithm

Step 1: Initially, $requests_u$ is the number of grants u requires in the wait-for graph to become unblocked.

Step 2: When $requests_u$ is (or becomes) 0, then u sends grant messages to all nodes in $Input_u$. When u receives a grant message, $requests_u \leftarrow, requests_u - 1$.

Step 3: If after termination of the deadlock detection run, *requests* > 0 at the initiator, then it is deadlocked (in the basic algorithm).

3.6 Mitchell and Merrit's Algorithm[7]

In 1984 Mitchell and Merrit's introduced an algorithm which comes under the category of edge-chasing algorithms. Here the deadlock detection is based on the nodes of *WFG* and each node has two labels: *private* and *public*. The unique label to that node is *private* label, and in the starting of the process the value of both *public* and *private labels* are same. Here the deadlock is detected when the *public* label nodes is propogated in the backward direction. When the transaction gets blocked then the value of *public* and *private* label in the WFG are increased from the previous value an also greater then the *public* label of blocking transaction. When the transaction receives its own *public* label the deadlock is detected. Therefore the backward propagation of largest *public* label is done in the deadlock cycle. Deadlock is easily resolved in this algorithm as the detection is done by only one process and is resolved by simply aborting it.

3.7 Singhal Algorithm [15]

In 1994 Singhal explained an algorithm based on P-out-of-Q model deadlock detection algorithms. It is called as a single phase algorithm and is based on the global state detection approach, which contains a set of messages outwards from an initiator process known as fan-out sweep and a set of messages inwards to the initiator process known as fan-in sweep. The traversal of the WFG in which all messages are sent against the direction of the *WFG* edges known as inward sweep or all the messages are sent in the direction of the *WFG* edges known as outward sweep. The algorithm records a snapshot of a distributed *WFG* in the outward sweep. Further, the recorded distributed *WFG* is reduced step by step to determine whether the initiator is deadlocked or not. The execution of both outward and the inward sweeps is done concurrently in the algorithm. There were complications in this algorithm as two sweeps were working concurrently so they can overlap each other at some common point or at same time. Therefore the reduction of the *WFG* should began before the process has completely recorded it.

There are n nodes in the system, and every pair of them is connected logically through a channel.

Lamport's clocks are used for assigning time-stamping events. The messages can be either *request, reply* or *cancel*. For the execution of p-out-of-q request, all the nodes receives a *request* message from the active node and is blocked until a sufûcient number of *reply* messages is received by him. When node *j* is blocked by node *i*, node *i* becomes a predecessor of node *j* and node *j* becomes a successor of node *i* in the WFG. The granting of a request is denoted by a *reply* message. When p out of its q requests have been granted node *i* is unblocked. When a node is unblocked, *cancel* messages are sent to withdraw the remaining q - p requests. Here the computation events are defined as sending and receiving of *request*, and *cancel* messages.

3.8 Chandy and Lamport Algorithm [9]

In 1985 Chandy, Lamport proposed an unique and simple technique, called as distributed snapshots technique, which was used for detecting stability in the distributed system: Here in this algorithm a local snapshot is taken by every process after recording the states of all channels which are incident upon it. All the previously taken local snapshots are assembled and collected to form a global snapshot of the distributed system, from which it is to be decided that whether the system has reached a stable state or not. For ensuring whether it is working correctly or not, a proposal was given by Chandy and Lamport which states that the processes should be coordinated in such a way that all local snapshots should lead to a meaningful resulting global snapshot.The algorithm relies on channels being first-in-first-out, and it requires O(IC I) control messages, where C is the set of channels in the system. This algorithm is known as message-efficient algorithm for the processes which takes the local snapshots. The channels are not required to be first-in first-out, and no control messages is required at all.

IV. DEADLOCK DETECTION ALGORITHMS BASED ON PRIORITIES

4.1 Ba Alom Algorithm [14]

In 2010 Alom proposed a deadlock removal and detection algorithm by using priorities. In this algorithm a table is made in which a list of all the transactions are noted along with their priorities. Here the deadlock is detected by drawing a wait for graph by the list of transactions using their priority. To make the system deadlock free, the transaction with the least priority is removed so that the resources occupied by that transactions are now free and are allotted to some other waiting transactions. There is also a drawback of this algorithm i.e. if there is some change in the priority of the deadlock then it may fail to detect any deadlock.

4.2 Michael Algorithm [5]

In 1971 Michael's proposed an algorithm which uses some new techniques in the algorithm: (1) Linear transaction structure *(LTS)* for each local site. (2) Distributed transaction structure (*DTS*): for global resource transaction communication. The maintainance of *LTS* is done at each site. *LTS creation*: If a data item that is held by a transaction T_q of same site and is requested by any transaction T_p then the value of p and q is stored to the *LTS*. *DTS creation*: All interconnected transactions are stored also the transactions intra request is recorded by *DTS* and managed by Data Manager (*DM*).

LTS of the site is checked for detecting the local deadlock. If the cycles are found then we use the priority technique in which priority of the involved transaction are entered by a transaction manager into a queue *Q*. Further the victim is chosen on the priority basis.

TABLE 1: Summary of Models Alongwith the Algorithm

S. No	Authors (Year of Publication)	Model	Algorithm	Use	Performance Analysis	Advantage	Disadvantage
1	Mithell and Meriits (1984)	Single Resource Model	Mithell and Merits	Network Model	It requires N (n-1)/2 trans-mission messages	Deadlock is easily detected using Private Label	It may detect false deadlock
2	Chandy and Mishra (1983)	And Model	Chandy and Mishra	Controlling sites	For N nodes it requires N(n-1) probes	The size of message is fixed as well as small	It may detect False feadlock
3	Chandy Mishra and Haas (1983)	Or Model	Chandy Mishra and Haas	Controlling sites	It exchanges e query and e reply messages where e= N (N-1), e=edges	The size of message is small as well as fixed	It may detect false deadlock
4	Bracha and Toueg's (1983)	$\binom{P}{Q}$ Model	Bracha and Toueg's	Networks	Uses snapshot to compute wait for graph	It uses a local snapshot to detect deadlock	It may detect false deadlock
5	Chandy and Lamport (1985)	Unrestricted Model	Chandy and Lamport	Control sites	Uses snapshots for checking the stability of deadlocks	The requires less number of messages to be transmitted	It may detect the false deadlock
6	Ho and Ramamoorthy (1982)		Ho and Ramamoorthy	Database sites	It requires 2N messages where N=sites	It double verifies the deadlock state	It may detect false deadlock
7	Obermarck (1982)	And Model	Obermarck	Network	It requires N(n-1) Messages to be Transmitted	It requires less messages to be transmitted in comparision to Ho's Algorithm	It may detect false deadlock
8	Alom (2010)		Alom	Priorities		Least priority edge is aborted	It may detect false deadlock
9	Michael's (1971)		Michael's	Sites		Deadlocked is indicated using priority of transaction	It may detect false deadlock
10	Singhal	(Pq) Model	Singhal	Nodes		Here the request contol and cancel messages are easily managed	It may detect false deadlock

V. CONCLUSION

Deadlock simply refers to a condition where the processes are waiting for each other to release the resource so that the process could occupy it and complete its pending work. it is common problem because the processes share common specific exclusive resource known as software lock or soft lock. Deadlock detection in distributed database has gone through extensive study. In this paper we studied a number of techniques used for detecting deadlock through various algorithms. [13] The large number of errors in published algorithms addressing the problem of distributed deadlock detection [Bracha and Toueg 1983; Chandy and Misra 1982; Ho and Ramamoorthy 1982; Obermarck 1982] shows that only rigorous proofs, using as little operational argumentation as possible, suffice to show the correctness of these algorithms. But if we go back to completely well-known and general principles like diffusing computations and global state detection, it is very much possible to achieve both elegance as well as correctness, even also for advanced models of deadlocks, without introducing unnecessary complexity. [11]

REFERENCES

[1] Chandy, K.M., Misra, J., and Haas, L. M. 1983. Distributed deadlock detection. ACM Trans. Comput. Syst. 1,2 (May), 144-156.

[2] R. Obermarck. "Distributed deadlock detection algorithm," ACM Trans. Database SW., vol. 7, pp. 187-208. June 1982. M

[3] B. M. M. Alom, F. Henskens, and M. Hannaford, "Deadlock Detection Views of Distributed Database," in International conference on Information Technology & New Generation (ITNG-2009) Las Vegas, USA: IEEE Computer Society, 2009.

[4] G. S. Ho and C. V. Ramamoorthy, "Protocols for Deadlock Detection in Distributed Database Systems" IEEE Transaction on Software Engineering, vol. 8:6, pp. 554-557, 1982.

[5] Elmagarmid A. K. A Survey of Distributed Deadlock Detection Algorithms, SIGMOD RECORD, Vol. 15, No.3, pp. 37-45, 1986.

[6] Mukesh Singhal "Deadlock Detection In Distributed Systems", November 1989

[7] D.P Mitchell and M.J. Merrit, "A Distributed Algorithm For Deadlock Detection and Resolution" Proc ACM Conf. Aug 1984

[8] Bracha, G., and Toueg, S. 1983. A distributed algorithm for generalized deadlock detection. Tech. Rep. TR 83-558, Cornell Univ., Ithaca, N.Y. BRACHA, G., AND TOUEG, S. 1984. A distributed algorithm for generalized deadlock detection. In Proceedings of the ACM Symposium on Principles of Distributed Computing (Vancouver, Canada, Aug.). ACM, New York, pp. 285-301.

[9] Chandy, K.M., and Lamport, L. 1985. Distributed snapshots: Determining global states of distributed systems. ACM Trans. Program. Lang. Syst. 3, 1 (Feb.), 63-75.

[10] Chandy, K.M., and Misra, J. Distributed computation on graphs: Shortest path algorithms. Commun. ACM 25, 11 (Nov. 1982), 833-837

[11] Edgar Knapp, "Deadlock Detection In Distributed Database Systems" ACM Computing surveys, Dec. 1987.

[12] Ajay Kshemkalyani and Mukesh Singhal, "Distributed Computing: Principles, Algorithms, and Systems" ACM Computing Machines, 2008

[13] Prabhsimran Singh, Sukhmanjit Kaur and Neha Bassan, "Approaches for Deadlock Detection for Distributed Systems", Dec. 2015

[14] Alom B.M. Monjurul, Frans Alexander Henskens, Michael Richard Hannaford, "Deadlock Detection Views of Distributed Database", IEEE Sixth International Conference on Information Technology: New Generations, Page(s):730–737, 2010

[15] Kshemkalyani, A. D., and Singhal, M., "Efficient Detection and Resolution of Generalized Distributed Deadlocks," *IEEE Trans. on Software Engineering*, January 1994

[16] Singhal, M., "Deadlock Detection in Distributed Systems," *IEEE Computer*, November 1989. ❐