Deadlock Detection and Recovery in Distributed Databases

Pooja Sapra
Research Scholar, MRIU

Suresh Kumar
FET, MRIU

R K Rathy
FET, MRIU

ABSTRACT
As the need of distributed processing increases, the complexity in handling of deadlocks also increases. In distributed databases, the conditions for the deadlocks are same as that in centralized but harder to detect, avoid and prevent. Therefore special procedures are required to resolve the deadlock. In this paper we propose a new distributed deadlock detection and recovery algorithm that not only detects deadlock but also resolve them efficiently by aborting less number of transactions. We also present comparative analysis of the proposed algorithm and observed that the proposed algorithm reduces the number of transactions that are to be aborted to resolve the deadlocks, thus improving the performance of the system.

General Terms
Databases, distributed, deadlocks, detection and resolution.

Keywords
Distributed databases, deadlock detection and recovery, transaction, wait-for-graph, transaction queue, linear transaction structure, distributed transaction structure, transaction manager.

1. INTRODUCTION:
A distributed system consists of a collection sites that are interconnected through a communication network. Each site has the local database and transactions running on them. Although the sites are dispersed, a distributed database system manages and controls the entire database as a single collection of data.

A deadlock is a situation in a system where transactions wait for one another [2] and none of them is able to proceed. In such situations, Deadlocks are generally depicted by wait-for graphs[12], which is a directed graph that indicates which transactions is waiting for which transaction for its completion. The graph consists of nodes and edges, where nodes of the graph represent transactions and edges of the graph represent the dependency among transactions. A direct edge from transaction T_i to transaction T_j is drawn, if the transaction T_i is waiting for a resource that is currently held by the transaction T_i. If the Wait-For-Graph contains a cycle then the system is assumed to be in a deadlock state. After the detection of deadlocks, their recovery is done. For recovery one of the transactions is considered as victim and aborted and then restarted.

In distributed systems, deadlock detection requires the local wait-for-graph and global-wait-for-graph to be constructed. A cycle in a LWFG indicates that a deadlock has occurred locally and a global deadlock is shown by GWFG. Even though there is no cycle in LWFG, it does not imply that no deadlock has occurred globally.

2. LITERATURE REVIEW :
There are two categories of distributed deadlock detection algorithms: Probe–based detection algorithms and edge chasing algorithms. Many authors proposed various algorithms under these categories, which are as follows:

Chandy et. al. [3], proposed an algorithm that uses transaction wait for graphs (TWFG) and probes to detect the local and global deadlocks respectively. It uses colored graphs for detecting the deadlocks and has the disadvantage of large space complexity and no deadlock resolution mechanism in order to make the system deadlock free.

Sinha et. al. [14], proposed an algorithm that was based on priorities of transactions to reduce the number of messages required for deadlock detection. In this scheme, a transaction's request for a lock on a data item is sent to the data manager for the item. When a transaction begins to wait for a lock, all the probes from its queue is propagated. When a data manager gets back the probe it initiated, deadlock is detected. Since the probe contains the priority of the youngest transaction in the cycle, the youngest transaction is aborted. In this algorithm data managers do not store probes and transactions are used as nodes of the graph. Due to this, another level of non-atomicity is added and complicated rules are required to add the new probes and delete the previous ones, whenever the WFG changes.

Obermack's Algorithm [12], builds and analyzes directed TWFG and uses a distinguished node at each site. The detection algorithm builds a TWFG and adds on all the information received from others processes also. Then it creates wait-for edges from external to each node representing agent of transaction that is expected to send on communication link and that is waiting to receive from communication link. Then it analyzes the TWFG and breaks down the youngest transaction creating the cycle. The algorithm does not work correctly because the WFG constructed at any instant does not represent a snapshot of the global WFG resulting into the detection of false deadlocks.

Ho's Algorithm [8], uses a resource table and transaction tables. Transaction table at each site maintains the information for resources held and waited for. The resource table at each site maintains information regarding the transactions holding and waiting for local resources. At the regular intervals, a site is chosen as a central controller which performs the deadlock detection. The drawback of this scheme is that it requires 4n messages, where n is the number of sites in the system.

Kawazu's Algorithm [9], algorithm works in 2 phases: in 1st phase it detects local cycles and in 2nd phase it detects global cycles. To detect the global deadlocks, the local wait for graphs are gathered to construct a pseudo wait for graph. This technique may suffer from phantom deadlocks.

3. DETECTION AND RESOLUTION OF DEADLOCKS IN DISTRIBUTED DATABASES:

The technique presented in [2] uses a greedy approach to find out the deadlocks and recovers them by aborting the youngest transactions. The algorithm works in the following steps:

- Create Linear transaction Structure (LTSi) for each local site i.
- Detect Local Deadlock cycle LDi.
- Create Transaction Queue TQi corresponding to each LDi.
- Abort the victim transaction.
- Create Distributed Transaction Structure (DTSi) for global communication.
- Detect Global Deadlock cycle GDi.
- Create Transaction Queue TQi corresponding to each GDi.
- Abort the victim transaction.

Interpretation: This technique assumes that the global deadlock detection is independent of local deadlock detection but there are some situations where it can be seen that this is not true.

The technique uses transaction queue to store the priority id for all transactions which are in local deadlock cycles or in global deadlock cycles, although assignment of priorities is random.

4. PROPOSED ALGORITHM:

In the proposed algorithm, we have modified the algorithm presented in [2], by relaxing the assumption, "global deadlock detection is independent of local deadlock detection". The proposed algorithm is as follows:

- Create Linear transaction Structure (LTSi) for each local site i.
- Create Distributed Transaction Structure (DTSi) for global communication.
- Detect Local Deadlock cycle LDi.
- Detect Global Deadlock cycle GDi.
- Find common request edge if exists.(CREi)
- Abort the transaction.
- Modify LTSi.
- Modify DTSi.
- Detect Local Deadlock cycle LDi.
- Detect Global Deadlock cycle GDi.
- Create Transaction Queue TQi corresponding to each LDi.
- Create Transaction Queue TQi corresponding to each GDi.

Abort the victim transaction

5. ILLUSTRATIONS:

5.1. Illustration 1:

Consider an example where we have taken two sites S1 and S2. Site S1 has transactions T1, T2, T3 and T4 and Site S2 has the transactions T5, T6 and T7. The Wait-for-graph for the transactions running on site S1 and S2 is depicted in Fig. 1.

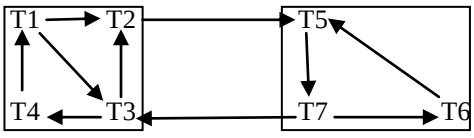

FIG. 1: WAIT FOR GRAPH

5.1.1. Deadlock detection by Alom et al.:

- Create Linear transaction Structure (LTSi) for each local site i.

Table1: Linear Transaction Structures at site 1 and 2.

LTS1:

p	q
1	2
1	3
3	2
3	4
4	1

LTS2:

p	q
5	7
7	6
6	5

- Detect Local Deadlock cycle LDi.
  LD1: {1→3, 3→4, 4→1}
  LD2: {5→7, 7→6, 6→5}
- Create Transaction Queue TQi corresponding to each LDi.

Table2: Transaction Queue for Local Deadlock Cycles 1 and 2.

TQ1:

Tno	TPid
1	1
3	2
4	3

TQ2:

Tno	TPid
5	1
7	2
6	3

- Abort the victim transactions T4 and T6.
- Create Distributed Transaction Structure (DTSi) for global deadlocks.

Table3: Distributed Transaction Structure (DTS)

p	q
2	5
5	7
7	3
3	2

- Detect Global Deadlock cycle GDi.
  GDC1: {2→5, 5→7, 7→3, 3→2}
- Create Transaction Queue TQi corresponding to each GDi.

Table4: Transaction Queue for Global Deadlock Cycle.

Tno	TPid
2	1
5	2
7	3
3	4

- Abort the victim transaction T3.

Total Number Of Transactions Aborted: 3(T3,T4,T6)

5.1.2 Deadlock detection by proposed algorithm:

- Create Linear transaction Structure (LTSi) for each local site i.

Table5: Linear Transaction Structures at site 1 and 2.

LTS1:

P	q
1	2
1	3
3	2
3	4
4	1

LTS2:

p	q
5	7
7	6
6	5

- Create Distributed Transaction Structure (DTSi) for global communication.

Table6: Distributed Transaction Structure (DTS)

p	q
2	5
5	7
7	3
3	2

- Detect Local Deadlock cycle LDi.
  LD1: {1→3, 3→4, 4→1}
  LD2: {5→7, 7→6, 6→5}
- Detect Global Deadlock cycle GDi.
  GDC1: {2→5, 5→7, 7→3, 3→2}
- Find common request edge if exists.(CREi)
  CRE: {5→7}
- Abort the transaction T5.
- Modify LTSi.

Table7: Linear Transaction Structures at site 1 and 2.

LTS1:

p	q
1	2
1	3
3	4
4	1

LTS2:

p	q
7	6
6	5

- Modify DTSi.

Table8: Distributed transaction Structure (DTS)

p	q
2	5
3	2
7	3

- Detect Local Deadlock cycle LDi.
  LD1: {1→3, 3→4, 4→1}
  LD2: NULL
- Detect Global Deadlock cycle GDCi.
  GDC1: NULL
- Create Transaction Queue TQi corresponding to each LDi.

Table9: Transaction Queue for Global Deadlock Cycle.

Tno	TPid
1	1
3	2
4	3

- Create Transaction Queue TQi corresponding to each GDi.
  NO CYCLE EXISTS.
- Abort the victim transaction T4.

So, total number of transactions aborted: 2 (T4, T2)

5.2. Illustration 2:

Consider the Wait-for-graph as shown in Fig. 3 for the transactions running on sites S1, S2 and S3.
Ti,j represents the transaction no. 'i' on site 'j'.
On site S1 the transactions T1, T2, T3, T4 are denoted as T1,1, T2,1, T3,1 and T4,1.
On site S2 transactions T3, T4, T5 and T6 are represented as T3,2, T4,2, T5,2, T6,2.
On site S3 transactions T7, T8, T9, T10, T11 are represented as T7,3, T8,3, T9,3,T10,3 AND T11,3.

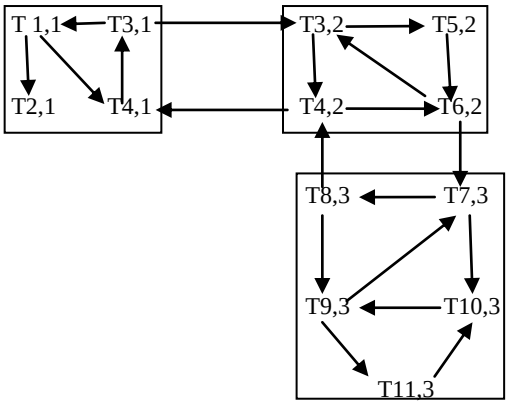

FIG. 3: WAIT-FOR-GRAPH

5.2.1. Deadlock detection by Alom et al.:

- Create Linear transaction Structure (LTSi) for each local site.

Table10: Linear transaction Structures at site 1, 2 and 3.

LTS1:

p	q
1,1	2,1
3,1	1,1
1,1	4,1
4,1	3,1

LTS2:

p	q
3,2	4,2
4,2	6,2
6,2	3,2
3,2	5,2
5,2	6,2

LTS3:

p	Q
7,3	8,3
8,3	9,3
9,3	7,3
7,3	10,3
10,3	9,3
9,3	11,3
11,3	10,3

- Detect Local Deadlock cycle LDi.
  LD1: {3,1→1,1; 1,1→4,1; 4,1→3,1}
  LD21: {3,2→4,2; 4,2→6,2; 6,2→3,2}

LD22: {5,2→6,2; 6,2→3,2; 3,2→5,2}
LD31: {7,3→8,3; 8,3→9,3; 9,3→7,3}
LD32: {7,3→10,3; 10,3→9,3; 9,3→7,3}
LD33: {10,3→9,3; 9,3→11,3; 11,3→10,3}

- Create Transaction Queue TQi corresponding to each LDi.

Table11: Transaction Queues for local deadlock cycles.

TQ1:

Tno	TPid
3,1	1
1,1	2
4,1	3

TQ21:

Tno	TPid
3,2	1
4,2	2
6,2	3

TQ22:

Tno	TPid
5,2	1
6,2	2
3,2	3

TQ31:

Tno	TPid
7,3	1
8,3	2
9,3	3

TQ32:

Tno	TPid
10,3	1
9,3	2
7,3	3

TQ33:

Tno	TPid
10,3	1
9,3	2
11,3	3

- Abort the victim transactions T4,1; T6,2; T3,2; T9,3; T7,3 and T11,3.
- Create Distributed Transaction Structure (DTSi) for global communication.

Table12: Distributed Transaction Structures (DTSi)

DTS1:

p	Q
3,1	3,2
3,2	4,2
4,2	4,1
4,1	3,1

DTS2:

p	q
4,2	6,2
6,2	7,3
7,3	8,3
8,3	4,2

- Detect Global Deadlock cycle GDi.
  GD1:{3,1→3,2; 3,2→4,2; 4,2→4,1; 4,1→3,1}
  GD2:{4,2→6,2; 6,2→7,3; 7,3→8,3; 8,3→4,2}
- Create Transaction Queue TQi corresponding to each GDi.

Table13: Transaction queues for global deadlock cycles.

Tno	TPid
3,2	1
4,2	2
4,1	3
3,1	4

Tno	TPid
4,2	1
6,2	2
7,3	3
8,3	4

- Abort the victim transaction T3,1 and T8,3.
  Total Number of Transactions Aborted: 8

5.2.2. Deadlock detection by proposed algorithm:

- Create Linear transaction Structure (LTSi) for Each site:

Table14: Linear Transaction Structures at site 1,2 and 3.

LTS1:

p	q
1,1	2,1
3,1	1,1
1,1	4,1
4,1	3,1

LTS2:

p	q
3,2	4,2
4,2	6,2
6,2	3,2
3,2	5,2
5,2	6,2

LTS3:

p	q
7,3	8,3
8,3	9,3
9,3	7.3
7,3	10,3
10,3	9,3
9,3	11,3
11,3	10,3

- Create distributed transaction structure for each site.

Table15: Distributed transaction Structures (DTSi)

DTS1:

p	q
3,1	3,2
3,2	4,2
4,2	4,1
4,1	3,1

DTS2:

p	q
4,2	6,2
6,2	7,3
7,3	8,3
8,3	4,2

- Detect Local Deadlock cycle LDi.
  LD1: {3,1→1,1; 1,1→4,1; 4,1→3,1}
  LD21: {3,2→4,2; 4,2→6,2; 6,2→3,2}
  LD22: {5,2→6,2; 6,2→3,2; 3,2→5,2}
  LD31: {7,3→8,3; 8,3→9,3; 9,3→7,3}
  LD32: {7,3→10,3; 10,3→9,3; 9,3→7,3}
  LD33: {10,3→9,3; 9,3→11,3; 11,3→10,3}
- Detect Global Deadlock cycle GDi.
  GD1:{3,1→3,2; 3,2→4,2; 4,2→4,1; 4,1→3,1}
  GD2:{4,2→6,2; 6,2→7,3; 7,3→8,3; 8,3→4,2}
- Find the CREi:
  CRE1: {4,1→3,1}
  CRE2: {4,2→6,2}
  Abort the edges: {4,1→3,1} and {4,2→6,2}
- Modify LTS:

Table16: Modified Linear Transaction Structures at site 1, 2 and 3.

LTS1:

p	q
1,1	2,1
3,1	1,1
1,1	4,1

LTS2:

p	q
3,2	4,2
6,2	3,2
3,2	5,2
5,2	6,2

LTS3:

p	q
7,3	8,3
8,3	9,3
9,3	7,3
7,3	10,3
10,3	9,3
9,3	11,3
11,3	10,3

- Detect Local Deadlock cycle LDi.
  LD1: null
  LD21: null
  LD22: {5,2→6,2; 6,2→3,2; 3,2→5,2}

LD31: {7,3→8,3; 8,3→9,3; 9,3→7,3}
LD32: {7,3→10,3; 10,3→9,3; 9,3→7,3}
LD33: {10,3→9,3; 9,3→11,3; 11,3→10,3}

- Create Transaction Queue TQi corresponding to each LDi.

Table17: Transaction Queues for local deadlock cycles.

TQ1:null TQ21:null

TQ22:

Tno	TPid
5,2	1
6,2	2
3,2	3

TQ31:

Tno	TPid
7,3	1
8,3	2
9,3	3

TQ32:

Tno	TPid
10,3	1
9,3	2
7,3	3

TQ33:

Tno	TPid
10,3	1
9,3	2
11,3	3

- Abort the victim transactions T3,2; T9,3; T7,3 and T11,3.
- Modify Distributed Transaction Structure (DTSi) for global communication.

Table18: Distributed transaction Structures (DTSi)

DTS1:

p	q
3,1	3,2
3,2	4,2
4,2	4,1

DTS2:

p	q
6,2	7,3
7,3	8,3
8,3	4,2

- Detect Global Deadlock cycle GDi.
  GD1: null
  GD2: null
- Create Transaction Queue TQi corresponding to each GDi.
  Null

Total Number of Transactions Aborted: 6

6. CONCLUSION:

Deadlock detection is the most important problem that must have a strong attention in case of distributed systems. Several algorithms have been proposed for detection and resolution of deadlocks. In this paper, we have analyzed the various algorithms and proposed a new technique for detection and recovery of deadlocks in distributed databases. Also we have analyzed the performance of proposed algorithm and compared with techniques presented in the literature. We observed that proposed technique resolve deadlock by terminating less number of transactions.

7. REFERENCES:

1) Alkhatib G. and Labban R. S., "Transaction Management in Distributed Database Systems: the Case of Oracle's Two-Phase Commit," The Journal of Information Systems Education, vol. 13:2, pp. 95-103, 1995

2) Alom B. M. M., Henskens F., Hannaford M., "Deadlock Detection Views of Distributed Database", Proc. of sixth International Conference on Information Technology: New Generations, pp. 730-737, 2009.

3) Chandy X. M. and Misra J.,"A Distributed Algorithm for Detecting Resource Deadlocks in Distributed Systems", Proc. of the First ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, New York, pp. 157-164, 1982.

4) Choudhary A. N., "Cost of Distributed Deadlock Detection: A Performance study", Proc. of Sixth International Conference on Data Engineering, Los Angeles, CA, pp.174-181, February, 1990.

5) Choudhury A. N., Kohler W. H., Stankovic J. A., and Towsley D.,"A Modified Priority Based Probe Algorithm for Distributed Deadlock Detection and Resolution," IEEE Transactions on Software Engineering, vol. 15:1, pp. 10-17, 1989.

6) Farajzadeh N. , Hashemzadeh M. ,Mousakhani M. and Haghighat A. T., "An Efficient Generalized Deadlock Detection and Resolution Algorithm in Distributed Systems," in International Conference on Computer and Information Technology, 2005.

7) Henskens F. and Ashton M. G. , "Graph-based Optimistic Transaction Management", Journal Of Object Technology, vol. 6: 6 pp. 131-148, 2007.

8) Ho G. S. and Ramamoorthy C. V., "Protocols for Deadlock Detection in Distributed Database Systems", IEEE Transaction on Software Engineering, vol. 8, no. 6, pp. 554-557, 1982.

9) Kawazu S., Susumu M., Menji I. and Kastumi T., "Two-Phase Deadlock Detection Algorithm in Distributed Databases", International Conference on Very Large Databases (VLDB) 1979 360-367.

10) Menasce D. A. and Muntz R. R., "Locking and Deadlock Detection in Distributed Databases", IEEE Transaction on Software Engineering, vol. 5, no. 3, pp. 195-202, 1979.

11) Mitchell D. P. and Merritt M. J., "A Distributed Algorithm for Deadlock Detection and Resolution", AT&T Bell Labs, Murray Hill, NJ 07974.

12) Obermack R., "Distributed deadlock Detection Algorithm", ACM Transaction on Database Systems, vol. 7, no. 2, pp. 144-56,1983.

13) Olson A. G. and Evans B. L., "Deadlock Detection for Distributed Process Networks", in ICASSP, 2005, pp. 73-76.

14) Sinha M. K. and Natarjan N., "A Priority Based Distributed Deadlock Detection Algorithm", IEEE Transaction on Software Engineering, vol. 11, no. 1, pp.67-80, 1985.

15) Srinivasan S. and Rajaram Ramaswamy, "An Efficient Detection and Resolution of Generalized Deadlocks in Distributed Systems", International Journal of Computer Applications, pp. 1-7, 2010.