

Deadlock Detection and Resolution in Distributed Database Environment

Abdullah Mohammed Rashid¹, Nor'ashikin Ali²

Basrah University, Basrah Iraq
College Of IT, University Tenaga Nasional Kajang, Selangor, Malaysia

Abstract: A distributed database system is a combination of sub-database separated over many sites communicated through a network. Deadlock is one of the most common problems that occur in distributed database implementation. Deadlock occurs when a multiple transaction locks the same data sources and every transaction waits for other to release. Deadlock detection and resolution is not easy in a distributed database system, because such system is composed of more than one site communicated to central database. The main objective of this study is to analyze several proposed algorithms to detect and resolve the deadlock in distributed database and propose a new technique to avoid the drawbacks of previous algorithms, besides enhancing the efficiency of detection and resolving the deadlock in distributed database.

Key Words: Distributed database, Deadlock, Detection and Resolving

I. Introduction

A distributed database environment is a combination of sub-database distributed over many sites communicated through a network (GROVER & KUMAR, 2012). According to Al-Murafi (2009), most organizations consider that the response time of the database is essential to achieve competitive advantage, such as for television companies. For example, the news programs are competing to obtain the most recent news early and broadcast early to satisfy the audience. Mitchell & Merritt (1984) mentioned that data retrieved from central database is the main bottleneck of time response exactly when the central database is distant from the organization. Thus, a distributed database enhances the organization's competitive advantage by increasing the efficiency and speed of data retrievals from different sites (Al-Murafi, 2009).

The database users interact with the database via transactions, which are defined as a set of logical units used to perform different tasks within the database, such as Read, Write, Update, Delete, and Insert (Jeffery, 2013). Concurrency problems occur when multiply users' access and update same source of data simultaneously (Himanshi Grover, 2013). According to Jeffery (2013), concurrency causes several drawbacks within the database, such as lost updates, uncommitted data and inconsistent retrievals.

The concurrency control is the processes of simultaneously managing the transaction and maintaining the database integrity without concurrency problems (Jeffery, 2013). The concurrency control uses two different techniques to prevent the concurrency, which are, (1) Locks: the transaction locks the data source needed in the current processes to prevent other transactions from accessing the same data through execution of transaction, and (2) Versioning: Database Management Systems (DBMS) provides a copy of the database with timestamp to every transaction that requires access to the database, then receives the update of database check if there is no conflict between multiple updates, the DBMS update central database, else based on timestamp, rejects the transaction, which causes the conflict (Jeffery, 2013).

Deadlock occurs when a transaction enters into wait state, for example multiple transaction locks the same data sources and every transaction waits for the other to release, when the requested data is not available because it is held by other transactions. There are three main approaches used to solve the deadlock, which are preventing the deadlock, avoiding the deadlock as well as detecting and resolving the deadlock (Chahar & Dala, 2013; Al-Murafi, 2009; Thakur & Deswal, 2014).

The main aim of this study is to review and analyze in detail some of the common algorithms proposed to detect and resolve the deadlock in distributed database. Furthermore, this study aims to determine the drawbacks of the proposed algorithms before providing recommendations that combine three algorithms together to avoid the individual weaknesses and enhance the efficiency of detecting and resolving the deadlock in distributed database systems.

II. Related Works

A. Related Deadlock Detection Techniques

There are too many studies conducted to detect and resolve the deadlock in distributed database. These studies address the deadlock by using three common techniques, which are prevention, avoidance or detection and resolution of the deadlock to increase the performance of distributed database.

Avoidance the deadlock : This technique handles the deadlock before it occurs. If the requested data sources are locked by other transactions, the current transaction has two choices: (1) wait for the locked data resource to be free, or (2) aborted (Swati Gupta, 2013; Thakur & Deswal, 2014; Bhatia & Verma, 2013).

Prevention of the deadlock: This technique stops the deadlock before it occurs. Each transaction identifies and blocks all data resources that will be used in transaction steps before executing the transaction (Swati Gupta, 2013; Thakur & Deswal, 2014; Bhatia & Verma, 2013). According to Jeffery (2013), preventing the deadlock is difficult to implement in real life because all programs are executed in a serial manner based on the demand routines, which means not all data sources needed in the transaction can be determined because some routines are not called yet.

Deadlock Detection: This technique detects the deadlock that has already occurred, by continuously scanning the transaction requests (subjects) and scanning the data sources (objects) to distinguish the deadlock. Then, one of the transactions detected as a victim to be aborted in terms of resolving the deadlock (Swati Gupta, 2013; Jeffery, 2013; Thakur & Deswal, 2014; Bhatia & Verma, 2013). There are several algorithms, techniques and procedures that are proposed to detect the deadlock and other criteria to detect the victim transaction in deadlock cycles. Table 1 illustrates some of these techniques and algorithms to detect the deadlock in distributed database, which is described in detail in the realistic example section 2.2 of the current study.

Table 1: Existing Techniques to Detect and Resolve the Deadlock

NO	Deadlock Detection Techniques	Description
1	B. M. Alom's techniques	Detect and resolve the deadlock by using LTS , DTS and priority tables
2	Brian M's algorithm	Detect and resolve the deadlock by using T-ID, Wait-For, Held-By, Request-Q and timestamp tables.
3	TWFG algorithm	Detect and resolve the deadlock by using Edges (E), Vertices (V) and timestamp tables
4	Path-pushing algorithms and Edge-Chasing algorithm	Detect and resolve the deadlock by using updated WFG tables and sending to neighbors

B. M. Alom (2009) has proposed a new algorithm to detect and resolve the deadlock in distributed database. The proposed algorithm is composed of two tables, which are local transaction structure (LTS) and distributed transaction structure (DTS) to record the transactions' request of the data resources, and other table to record the priority of transactions. When the deadlock occurs and detected , the algorithm is detect the victim node based on the priority table, where the lowest priority transaction is aborted and all data resources held by the aborted transaction are granted to other transactions in terms of resolving the deadlock as will described in the realistic example section.

Brian M's .Johnston et al. (1991) have proposed new algorithm to detect and resolve the deadlock in distributed database. The proposed algorithm has the main transaction table structure, which consists of T-ID ,Wait-for, Held-by, and Request-Q; when intersection occurs between the wait-for and requested –Q means there is deadlock. The victim transaction is allocated based on the timestamp of the younger transaction that will be aborted because it has a few requested data resources as will described in the realistic example section.

Mitchell & Merritt (1984), Chahar & Dala, (2013) and Swati Gupta (2013) have proposed Transaction-Wait-For-Graph algorithm to detect and resolve the deadlock in distributed database. The proposed algorithm composed from Edges (E) and Vertices (V) in current algorithm, where E represents transactions and V represents the data resources. The deadlock occurs when the graph contains a cycle of transaction that requests a resource held by them. The victim transaction is detected based on the timestamp of the younger transaction that will be aborted because it has a few requested data resources as describe in the realistic example section 2.2.

Bhatia & Verma (2013) have proposed a new algorithm to detect and resolve the deadlock in distributed database. The proposed algorithm uses WFG individually for each site of distributed database. When WFG allocates the deadlock, every sit updates its own WFG and sends to a number of neighbors. This process continues until one of the sits complete the whole picture about the global deadlock then try to solve the deadlock by one of three mechanisms, which are resolve by Preemption, resolve by Rollback, and resolve by Termination, as will described theoretically in the realistic example section.

B. Realistic Example of Deadlock Detection and Resolutions

This section applies and analysis the proposed algorithms in practical example of distributed database, only last Path-Pushing algorithms will discusses theoretically. Suppose the distributed database system (DDS) consist from two sits site1 and site2 , site 1

maintaining four transactions which are t1,t2,t3, and t4 as well as sit2 maintaining other four transactions which are t5,t6,t7, and t8 as shown in Figure 1.

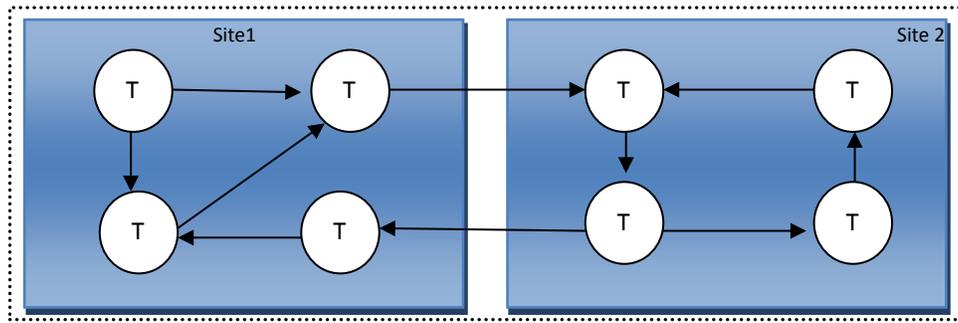


Figure 1: Example of distributed database structure composed of two sites

• **First over all analyzing the B. M. Alom’ algorithm based on the example present in figure 1:**

This algorithm consist from three tables which are (1) local transaction structure (LTS) is a tables recorded the locals transactions’ requests individually for each site as shown in table 2 for site1 and table 3 for site 2, (2) Distributed transaction structure (DTS) is a table to recorded the global transactions ‘requests which are participate more than one site as shown in table 4, and (3) transaction query is a table maintain the transactions ID related to their priorities as shown in table 5 and 6 related to site 1 and site 2 gradually and table 7 shows the transaction priority for DTS .

T-ID	Request -At
T1	2
T1	3
T2	3
T4	2

Table 2: Site 1 Transactions’ request

T-ID	Request -At
T5	6
T6	8
T8	7
T7	5

Table 3: Site 2 Transactions’ request

T-ID	Request -At
T1	2
T1	3
T2	3
T3	5
T5	6
T6	4
T4	2
T8	7
T7	5
T6	8

Table 4: DTS for site1 and site 2

T-ID	T-Priority
T1	1
T2	2
T3	3
T4	4

Table 5: Site 1 Transactions’ Priority

T-ID	T-Priority
T5	3
T6	1
T7	4
T8	2

Table 6: Site 2 Transactions’ Priority

T-ID	T-Priority
T1	1
T5	2
T3	3
T7	4
T2	5
T8	6
T4	7
T6	8

Table 7: DTS Priority

Detecting the deadlock is through scanning the transaction requests looking for any cycles. Table 2 illustrates that, there is no cycle between transactions’ request that mean there is no deadlock occurs, can say that site 1 is deadlock free. Table 3 illustrates that there is a clear cycle between transaction requests in site2, thus there is a deadlock. Table 6 shows that, T6 has lowest priority in site 2, thus T6 will be aborted because it is youngest transaction and has low requested resource. Table 4 illustrates that, there is a cycle between transaction’s requests from site1 and site 2, thus there is a deadlock occurs between transaction’s requests. Table 7 shows that T5 is lower priority transaction has participates in global deadlock. Thus, T5 has been aborted because it is youngest transaction

in DTS caused the deadlock. Figure 2 show the distributed database without any deadlocks between transactions' requests based on B. M. Alom' algorithm .

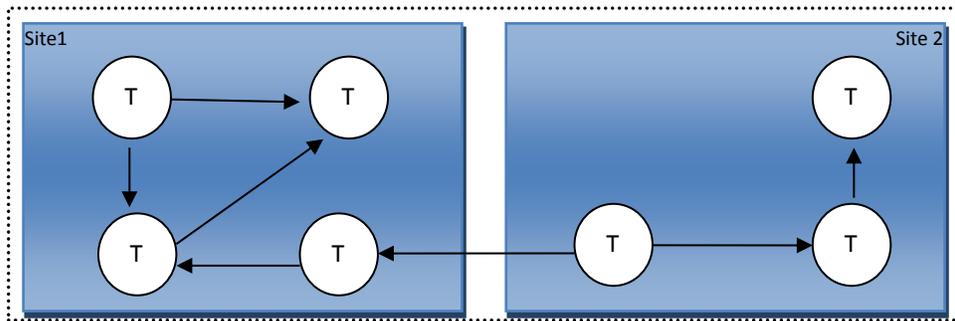


Figure 2: Distributed database without deadlocks

• **Second, analyzing the Brian M. Johnston's algorithm based on the example present in figure 1:**

This algorithm considers that, each transaction has T-ID represent the transaction identification, and TS represent the timestamp associate with T-ID which presents when the transaction join the execution stage . From other hand, there are three main variables for each transaction which are (1) Wait-for, (2) Held-by, and (3) Request-Q. If the T_i is not waiting for any other transaction then $Wait\text{-}for(T_i)$ is set to nil, otherwise, $Wait\text{-}for(T_i)$ refer to which transaction is at the head of the locked data object. $Held\text{-}by(T_i)$ is set to nil if the current transaction is executing, else it stores the transaction that is holding the data object required by the current transaction. $Request\text{-}Q(T_i)$ maintains all pending requests for data resources that are holed other transaction. Each element in the $Request\text{-}Q(T_i)$ is a pair (T_j, D_i) , where T_j is the requesting transaction and D_i is the particular data source held by T_i .

When transaction T_i establish data request for to lock D_j , if D_j free can granted directly to T_i and locks under T_i identifications, else several processes required to perform which are :

- D_j sends a not granted message to T_i along with the transaction identifier locking D_j
- T_i joins the $Request\text{-}Q(T_j)$ and sets its $Wait\text{-}for$ equal to $wait\text{-}for(T_i)$.
- Now T_i initiates a update message to modify all the $Wait\text{-}for$ variables which are affected by the changes in $Locked$ by variable of the data objects.
- Update message is a recursive function call that will continue updating all elements of every $Request\text{-}Q$ in the chain.
- When a transaction T_j receives the update message it checks if its $Wait\text{-}for$ value is the same as the new $Wait\text{-}for$ value.
- This message continues until T_i discovers that $Wait\text{-}for(T_i)$ intersected with $Request\text{-}Q(T_i)$ is not nil.
- Thus, the deadlock occurs then should abort the T_i and granted its data sources to other transaction required it.

Execute this algorithm by based on the example in figure 1. Table 8 and table 9 show the transaction structure for site 1 and site 2 .Finally, table 10 shows the transaction structure in distributed environment both site1 and site 2.

T-ID	TS	Wait-for	Held-By	Request-Q
T1	2	T2,T3	T2,T3	Nil
T2	1	T2	T3	T1,T4
T3	3	Nil	Nil	T1,T2
T4	4	T2	T2	Nil

Table 8: Transaction Structure of site1

T-ID	TS	Wait-for	Held-By	Request-Q
T5	3	T6	T6	T7
T6	1	T6	T8	T5
T7	2	T6	T5	T8
T8	4	T6	T7	T6

Table 9: Transaction Structure of site1

$Wait\text{-}for(T1) \cap Request\text{-}Q(T1) = nil$
 $Wait\text{-}for(T2) \cap Request\text{-}Q(T2) = nil$

$Wait\text{-}for(T5) \cap Request\text{-}Q(T5) = nil$
 $Wait\text{-}for(T6) \cap Request\text{-}Q(T6) = nil$
 $Wait\text{-}for(T7) \cap Request\text{-}Q(T7) = nil$

Wait-for (T3) \cap Request-Q (T3) = nil
 Wait-for (T4) \cap Request-Q (T4) = nil

Wait-for (T8) \cap Request-Q (T8) = T6

Based on reviews the intersection values of table 8 , there is no deadlock occur because there is no intersection exist between Wait – for(Ti) and request-Q(Ti) . The intersection values of table 9 shows that, there is a deadlock occur because Wait-for(T8) \cap Request-Q (T8) = T6 , thus, one of these transactions T8 or T6 should be aborted based on lower timestamp. Therefore, the timestamp of T6 is 4 which is less than the time stamp of T8, thus T8 should be abort to resolve the deadlocks.

T-ID	TS	Wait-for	Held-By	Request-Q
T1	7	T2	T2,T3	Nil
T2	1	T2	T3	T1,T4
T3	5	T2	T5	T1,T2
T4	2	T2	T2	T6
T5	6	T6	T6	T7,T3
T6	3	T6	T4,T8	T5
T7	8	T6	T5	T8
T8	4	T6	T7	T6

Table 10: Transaction Structure of site1and site 2

The intersections review based on table 10 shows that, there in two deadlocks occur first one caused by transaction 3 and second one caused by transaction 8 as describe in the following intersection formulates.

Wait-for (T0) \cap Request-Q (T0) = nil
 Wait-for(T1) \cap Request-Q (T1) = nil
 Wait-for (T2) \cap Request-Q (T2) = nil
Wait-for(T3) \cap Request-Q (T3) = T2
 Wait-for(T5) \cap Request-Q (T5) = nil
 Wait-for(T6) \cap Request-Q (T6) = nil
 Wait-for(T7) \cap Request-Q (T7) = nil
Wait-for(T8) \cap Request-Q (T8) = T6

To resolve the deadlock required to select the victims transactions based on time stamp values. $TS(T2) < TS(T3)$, thus T3 should be aborted to resolve first deadlock , and $TS(T8) > TS(T6)$, thus T8 should be aborted to resolve the second deadlock as shown in figure 3.

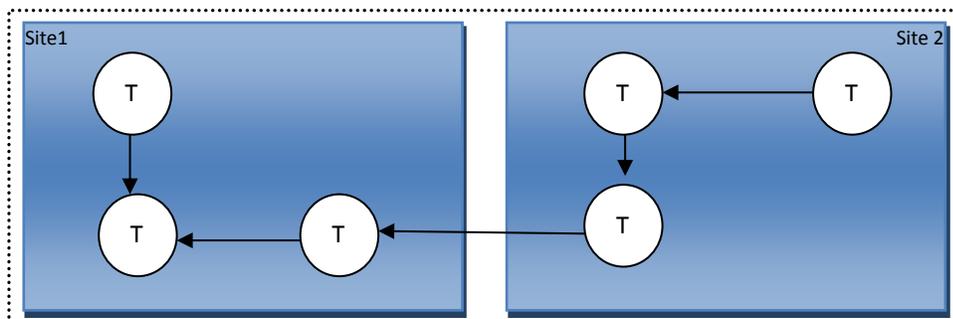


Figure 3: Distributed database without deadlocks

- **Third analyzing the TWFG algorithm which is proposed by Swati Gupta (2013), based on the example present in figure 1:**

This algorithm consists from two tables which are local transaction structure (LTS) maintains the local transactions' requests individually for each site and distributed transaction structure (DTS) stores the global transactions 'requests which are participate more than one site. LTS and DTS are created based on the transactions' timestamp. This algorithm will analysis based on the example present in figure 1 by executes the following steps.

1. All transaction T_i request data sources maintaining in array namely P and the data source have been needed stores in array called Q, array vector used to keep the scan values though execution this algorithm initially it is empty, stack to store the matching values, and finally temp variable used to store the searching values.
 2. First transaction from array P enters to vector array.
 3. The corresponding data sources from array Q is store in temp variable.
 4. Search the temp variable in array of transaction P, every matching value as a result of searching passed to stack.
 5. Repeat until complete array P is empty: Take the most top value in the stack, only the Q value then searching in vector array for matching results, if the results is false there is no matching values enter the value of P to the vector array and go to step 3. Else if the result is true that mean there is a deadlock occur. The victim transaction will be allocated and aborted based timestamp value resolve the problem of gridlock.
 6. For unsuccessful search if temp is not found in array P then make the stack empty.
- Finally, to detect the deadlock in global sites: create the DTS table to maintain the transaction request from different sites, implement the local deadlock detection algorithm in each site individually to resolve the global deadlock cycle.

Table 11 illustrates the transactions timestamp for site1 and site 2, Table 12 shows the LTS for site1 and site 2, and figure 4 and figure 5 explain the algorithm execution steps in site 1 and site 2.

Site1		Site2	
T-ID	TS	T-ID	TS
T2	1	T6	3
T4	2	T8	4
T3	5	T5	6
T1	7	T7	8

Table 11: Timestamp for site1 and site2

Site1		Site2	
P	Q	P	Q
2	3	6	8
4	2	8	7
1	3	5	6
1	2	7	5

Table 12: LTS for site1 and site2

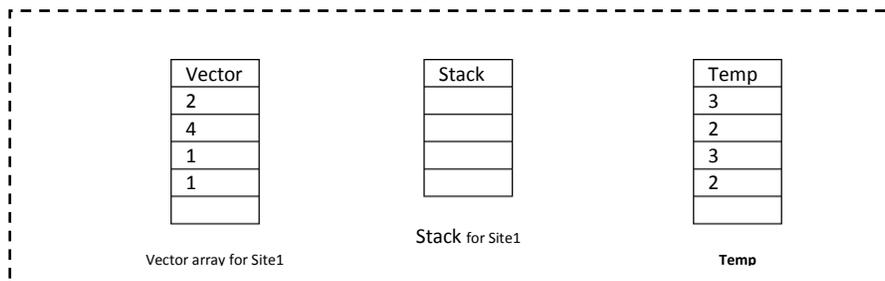


Figure 4: Deadlock detection in site 1

According to vector array and stack directions there is no matching occur between pairs (P, Q). Thus, there is no deadlock occur in site 1.

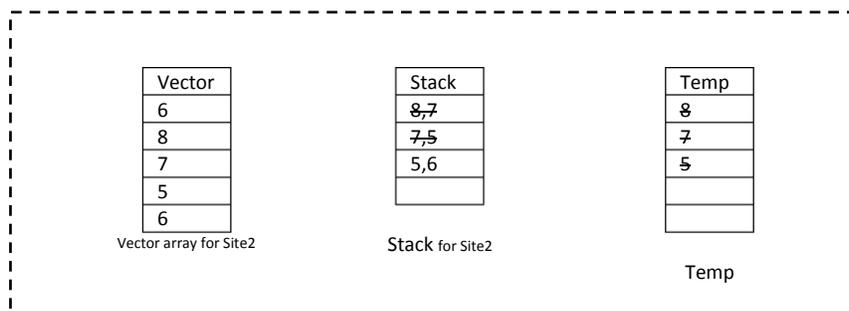


Figure 5: Deadlock detection in site 2

According to vector array of site 2 there is matching occur for the transaction from node 5 to node 6. Thus, the transaction with lowest timestamp should be aborted to resolve this problem. According to table 11, the $TS(T5) > TS(T6)$, thus, T5 will be aborted. Table 13 shows that, the LTS after detect and resolve the deadlock.

Site1		Site2	
P	Q	P	Q
2	3	6	8
4	2	8	7
1	3		
1	2		

Table 13: LTS for site1 and site2

Regarding to deadlock detection in distributed database table 15, illustrates the timestamp for site 1 and site 2 transactions that have participate in the cycle between two sites. Table 14 shows the DTS for site1 and site 2. Now, to detect the deadlock and victim transaction should use the P, Q, vector, stack and temp variable as shows in figure 6.

Site1&2	
P	Q
2	3
4	2
6	4
3	5
5	6

Table 14: DTS for both site1 and site2

Site 1 & 2	
T-ID	TS
2	1
4	2
6	3
3	5
5	6

Table 15: timestamp for site1 and site2

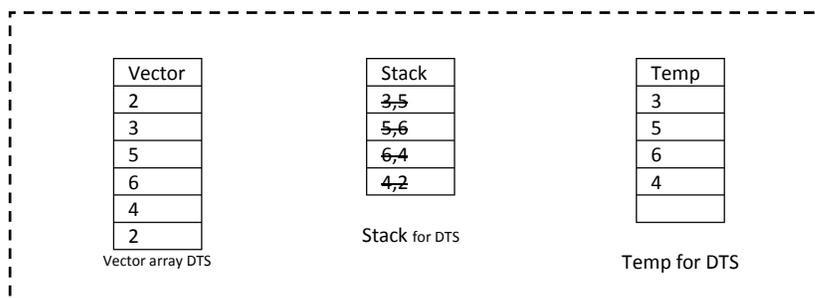


Figure 6: Deadlock detection in DTS

According to array vector values, there is a matching value between T2 and T4, thus the deadlock occur between T2 and T4. One of these two transaction should be aborted $TS(T4) > TS(T2)$, thus T4 will be aborted to resolve the deadlock problem and continuous without gridlock. Finally, figure 7 illustrates the distributed database without deadlock.

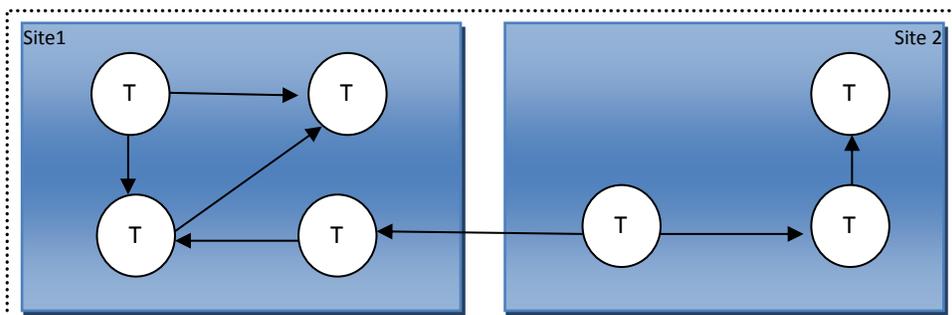


Figure 7: Distributed database without deadlock

- **Furth , Analyzing Path-pushing algorithms theoretically which is proposed by Bhatia & Verma (2013)**

Path-pushing algorithms: The importance of this technique is creating a Global-Wait-for-Graph (GWFG) by depending on Local-Wait-For-Graph (LWFG). Every site sends its own LWFG to all neighbors' sites to detect the deadlock. The sites modified their LWFG based on shared information from neighbors. This process continues until some of the sites have completed a whole picture of the distributed database transaction. Finally, it can decide if there is existing deadlock or not. This algorithm is called path-pushing algorithms because it keeps track of the path along the sites to detect the transaction that causes the gridlock. On the other hand, edge-chasing algorithm is described as a deadlock in distributed database that can be detected by sending a message called probes along the edges of the graph. When the initiator probe receives a matching probe, it knows there is a cycle in the graph (Deadlock). To resolve the deadlock, there are three main approaches that can be used to perform this task, which are, 1) Resolve through Preemption, 2) Resolve through Rollback, and 3) Resolve through Termination.

III. Discussion

B. M. Alom (2009) algorithm detects the deadlock cycle in site 1 and site 2 individually then in distributed database and solves the deadlock by using priority values. In site 1, there was no deadlock, but in site 2 and distributed database, there was a deadlock. The victim nodes are selected based on priorities of transactions. Transaction 6 was the main node that caused the deadlock in site 2 while transaction 5 was the main node that caused the deadlock in distributed database based on priority schedule. The main drawback of this algorithm is the deadlock detection depends only for the transaction priorities, if there is a change in the transactions priorities, the algorithm may fail to detect the right deadlock.

Brian M's algorithm allocates the deadlock in local and distributed database and resolves the deadlock by using timestamp values. The deadlock occurs in site 2 and distributed database while there is no deadlock in site 1. In terms of resolving the deadlock, the lowest timestamp transaction will be aborted to avoid the deadlock from occurring. Based on the timestamp schedule in site 2, transaction 8 was aborted and in distributed database, transaction 3 and transaction 8 were aborted to avoid the deadlock. The number of transactions is aborted by Brian M's algorithm in distributed database more than the number of transaction aborted by Alom's algorithm, which is only transaction 6. The main limitation of this algorithm is that there are no criteria of deciding which transaction needs to be aborted early to reduce the repeated time of detections.

TWFG's algorithm detects and resolves the deadlock in local and global database environments. There is no deadlock detection in site 1. However, site 2 and distributed database contains a deadlock. To resolve the deadlock, the victim transaction, which is the youngest transaction based on time stamp values, should be aborted. Transaction 5 was the main transaction causing the deadlock in site 2 and distributed database, thus it is selected to be aborted as the youngest transaction. The main problem of this algorithm is starvation; this algorithm often aborts the youngest transaction that causes the critical starve to the youngest transaction.

Finally , all the proposed algorithm detect and resolve the deadlock in distributed database, but all of them have some limitations such as priority, standard criteria and starvation. This justifies the need to create new techniques to remove the drawbacks of the proposed algorithms.

IV. Recommendation

The main drawback of B. M. Alom's algorithm is detecting the deadlock by using transaction priorities schedule, when a change in the priorities would result in this technique possibly failing to detect the deadlock. The main drawback of Brian M's algorithm is that there are no standard criteria of allocating which transaction should to be aborted early to reduce the repeated time of detections. Meanwhile, the main drawback of TWFG algorithm is starvation. This algorithm often aborts the youngest transaction that causes the critical starve to the youngest transaction. We can suggest new techniques that composed of B. M. Alom, TWFG and Brian M's algorithms implemented together to solve these three weaknesses as the follows.

1. Detect the deadlock by using B. M. Alom's algorithm.

2. Use Brian M's algorithm to allocate the victim transaction to solve the changes of priorities problem based on timestamp.
3. Use flag variable associated with each transaction. Initially set all flags as Zero value (flag=0), which means not aborted yet.
4. When detecting the victim transaction as a younger transaction based on step 2, check if flag=0; this means that the transaction can be aborted and set the flag value as one (flag=1), which is not allowed to abort the youngest transaction and can abort the next earlier timestamp transaction.

This procedure detects the deadlock by using B. M. Alom's algorithm, resolve the deadlock by using Brian M's algorithm and solve the starvation drawback by using flag variable. Thus, the proposed algorithm detects and resolves the deadlock efficiently.

V. Conclusion

The in-depth reviews of common algorithms which are used to detect and resolve the deadlock in distributed database show that there is no complete algorithm to detect and resolve without any negative effect, such as first algorithm facing drawback when the priorities of the transaction are changed; secondly algorithm faced a limitation where there is no standard criteria to resolve the deadlocks, and thirdly the algorithm has real weaknesses, which are starvations. This research proposed new techniques to detect and resolve the deadlock by using the reviewed algorithms' concepts together and add flag values as shown in the recommendation section.

References

al-murafi, w. s. h. *database fragmentation technique for news websites (doctoral dissertation, middle east university)*

alom, b. m., henskens, f., & hannafor, m. (2009, april). deadlock detection views of distributed database. in *information technology: new generations, 2009. itng'09. sixth international conference on* (pp. 730-737). ieee

bhatia, y., & verma, s. (2014). deadlocks in distributed systems. *international journal of research*, 1(9), 1249-1252

chahar, p., & dalal, s. deadlock resolution techniques: an overview. *international journal of scientific and research publications*, 56

jeffrey hoffer v ramesh heikki topi isbn-10: 0273779281 • isbn-13: 9780273779285 ©2013 • pearson • paper, 628 pp published 12 oct 2012 • reprinting - limited stock. add to basket to check current availability.

- see more at: <http://catalogue.pearsoned.co.uk/catalog/academic/product?isbn=9780273779285#downloadable-instructor-resources>

grover, h. (2013, february). a distributed algorithm for resource deadlock detection using time stamping. in *international journal of engineering research and technology* (vol. 2, no. 11 (november-2013)). esrsa publications

grover, h., & kumar, s. (2012). analysis of deadlock detection and resolution techniques in distributed database environment. *international journal of computer engineering & science*, 2(1), 17-25

gupta, s. (2013). deadlock detection techniques in distributed database system. *international journal of computer applications*, 74(21), 41-45

johnston, b. m., javagal, r. d., datta, a. k., & ghosh, s. (1991, march). a distributed algorithm for resource deadlock detection. in *computers and communications, 1991. conference proceedings., tenth annual international phoenix conference on* (pp. 252-256). ieee.

mittchell, d. p., & merritt, m. j. (1984, august). a distributed algorithm for deadlock detection and resolution. in *proceedings of the third annual acm symposium on principles of distributed computing* (pp. 282-284). acm

thakur, s., & deswal, k. (2014). analysis for deadlock detection and resolution techniques in distributed database. *international journal of research*, 1(9), 1312-1316

AUTHORS

First Author – Abdullah Mohammed Rashid, Master student, College Of IT University Tenaga Nasional, Kajang, Selangor, Malaysia.

Email : Abdall_rshd@yahoo.com

Second Author – Nor'ashikin Ali, PHD, College Of IT, University Tenaga Nasional, Kajang, Selangor, Malaysia,

Email:shikin.uniten@gmail.com