2011 International Conference on Computer Communication and Management
Proc .of CSIT vol.5 (2011) © (2011) IACSIT Press, Singapore

Deadlock Detection and Resolution in Neighbour Replication on Grid

Noriyani Mohd Zin, A.Noraziah, A.H. Beg, Ainul Azila Che Fauzi

Faculty of Computer System & Software Engineering, Universiti Malaysia Pahang, Pahang, Malaysia

Abstract. Distributed database system has a collection of sites that will increase the data availability, reliability of data and the execution speed in less time. However, the management on transaction is important in managing deadlock problems. In this paper presents a new algorithm to manage transaction on neigbour replication in distributed database system. The Neighbour Replication on Grid (NRG) Deadlock Detection (NRGDD) is the algorithm developed based on the Multi-Cycle Deadlock Detection and Recovery (MC2DR) approach. This algorithm will detect and resolve the deadlock problem that occurs during the transaction. Each node of the different set of transactions will communicate by passing the probe message to each other. To resolve deadlock problem only one node that will be detect as a victim that cause of deadlock occur and this node will be abort and release that site to another transaction to obtain the lock. Finally, this algorithm successfully implemented on NRG model that can detect and resolve the deadlock problems.

Keywords: Distributed Databases, Replication, NRG Replication Model, Deadlock Detection and Resolution.

1. Introduction

In distributed database system has a collection of sites that interconnected via communication network. Through this system it will increase the data availability, reliability of data and increased the execution speed in less time. A distributed database system allows applications to access data from local and remote databases [1]. In this system, several characteristics are considered such as: (1) provides an interface to user which is transparent to where the data actually resides; (2) ability to locate the data; (3) network-wide concurrency control and recovery procedures; (4) translation of queries and data between heterogeneous systems [2]. In this system, to access data item simultaneously must be synchronous to preserve the data consistency. Replication is a technique use in distributed database that will replicate the data in several sites. If one of the nodes has failed, it will failed independently and not affect to others node. Consequently, the data replication will improve the reliability, availability and performance of the data. Replication in distributed environment receives particular attention for providing efficient access to data, fault tolerance and enhance the performance of the system [3-5].

In order to manage the transaction management, the concurrency control and deadlock detection is the most important problem that must have a powerful attention when sharing in distributed systems [6]. The lock mechanism is use when the transaction make request to get a data. If the data is available, the transaction that make a request will get a lock for that data, otherwise it will wait until the data is unlock or released then it can be acquired again. In this situation, a deadlock may occur in which every transaction involve in the deadlock are waiting to grant the data that has been lock by other transaction that make a circular wait until an action is taken to detect and resolve deadlock problems.

Deadlock detection is very difficult in a distributed database system because no controller has complete and current information about the system and data dependencies [8]. The proposed algorithm in [8] does not detect any false deadlock or exclude any really existing deadlocks and in this technique global deadlock is not dependent on the local deadlock. It based on creating Linear Transaction Structure (LTS), Distributed Transaction Structure (DTS) and local global abortion.

In [9] has proposed an efficient deadlock detection (EDD) algorithm that detect deadlock based on based on threads, processes which are acquired, released or stopped and which thread wait for the other and causes the deadlock [9]. The structure that represents the relation between resources and threads (processes) for faster and accurate detection has been introduce in this paper for efficient deadlock detection. This algorithm can

+ Noriyani Mohd Zin. Tel.: +6095492121; fax: +6095492144.
E-mail address: noriyanimz@gmail.com.

detect only the present of real deadlocks that will coordinate the share resources in order to use the minimum time to detect deadlocks [9]. In [7] proposed multi-cycle deadlock detection and recovery (MC2DR) algorithm contributes that its (i) can detect all deadlocks reachable from the initiator of the algorithm in single execution, even though the initiator does not belong to any deadlock, (ii) it can detect multi-cycle deadlocks i.e., deadlocks where a single process is involved in many deadlock cycles, (iii) it decrease the deadlock detection algorithm initiations, phantom deadlock detections, deadlock detection duration and the number of useless messages and (iv) it provides with an efficient deadlock resolution methods.

In this paper, the new algorithm will be produced based on the MC2DR algorithm that will expand in the Neighbour Replication on Grid (NRG) replication model. NRG has been proposed in our previous work [10]. NRG is treading a new path in replication that helps to maximize the write availability with low communication cost due to the minimum number of quorum size required. The purpose of this research is to show how the new algorithm can detect the existence of real deadlock and resolve it.

The rest of the paper is organized as follows. In Section 2, presents deadlock detection and resolution transaction model. The implementation processing and expected outcomes will be shows in Section 3. Section 4 concludes the paper and Section 5 is acknowledgement.

2. Transaction Model

The transaction become stuck because of the deadlock has occur when different set of transaction waiting for each other to obtain the same resource. In this section will present NRG deadlock detection (NRGDD) transaction model.

A. NRGDD Transaction Model Notation

In this section, we defined the following notations:

a) T is a transaction
b) D is the union of all data object manages by all transaction T of NRG and x represents one data object (or data file) in D to be modified by an element of T_α, T_β, T_γ, T_δ, and T_θ.
c) $\lambda = \alpha, \beta, \gamma, \delta, \theta$ where it represent different group for the transaction T.
d) PM is a probe message. It contain a set of probe messages where $PM = \{InitID, VictimID, DepCnt, RouteString\}$. See Table 1.

Table 1: Probe Message.

Probe Message	Descriptions
InitID	Contains the identity of initiator of the algorithm
VictimID	A node or transaction that detects the deadlock sends the victim message to the node or transaction that cause of deadlock occurs. This node will be victimized for deadlock resolution.
DepCnt	The number of successor represent as a node or transaction which is waiting for resource.
RouteString	The node or transaction IDs visited by another node's (transaction's) probe message in order.

e) NRG transaction elements $T_\alpha = \{T_{\alpha x,PM} \mid PM=InitID, VictimID, DepCnt, RouteString\}$ where $T_{\alpha x,PM}$ is a probe message elements of T_α transaction.
f) NRG transaction elements $T_\beta = \{T_{\beta x,PM} \mid PM=InitID, VictimID, DepCnt, RouteString\}$ where $T_{\beta x,PM}$ is a probe message elements of T_β transaction.
g) NRG transaction elements $T_\gamma = \{T_{\gamma x,PM} \mid PM=InitID, VictimID, DepCnt, RouteString\}$ where $T_{\gamma x,PM}$ is a probe message elements of T_γ transaction.
h) NRG transaction elements $T_\delta = \{T_{\delta x,PM} \mid PM=InitID, VictimID, DepCnt, RouteString\}$ where $T_{\delta x,PM}$ is a probe message elements of T_δ transaction.
i) NRG transaction elements $T_\theta = \{T_{\theta x,PM} \mid PM=InitID, VictimID, DepCnt, RouteString\}$ where $T_{\theta x,PM}$ is a probe message elements of T_θ transaction.
j) Each node or transaction has a probe message storage structure also known as *ProbeS*, at most one probe message will be store on *ProbeS* at particular time. The history of *ProbeS* is independent; when the deadlock has been detected the probe message is erased from *ProbeS*.
k) Transaction $T_{\lambda x,PM}$ that detects the deadlock send a victim message to the transaction found to be victimized for the deadlock resolution. Victim message will be used for deleting probes from respective storage entries.

NRGDD transaction model consider different set of transactions T_α, T_β, T_γ, T_δ, *and* T_θ. All elements T_α, T_β, T_γ, T_δ, *and* T_θ may request data object x simultaneously at any site of $S(B)$ either at the same or different site.

Each set of transactions communicate with each other by message passing. Each of them bring the elements of probe message or *PM* where *PM*={*InitID, VictimID, DepCnt, RouteString*}. At most one probe message will be store in probe storage, *ProbeS*.

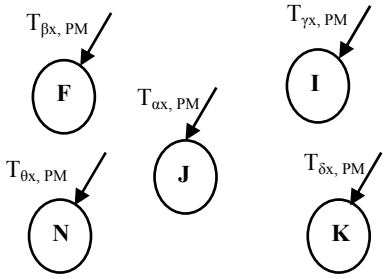

Fig. 1: Different set of transaction request a different site.

An Illustration Example: Let us illustrate the working of new algorithm for detecting deadlock, through an example. Consider the situation shown in Fig. 1. A different set of transactions T_α, T_β, T_γ, T_δ, *and* T_θ request a lock from a set of sited where $S(B_x) = \{J, F, I, K, N\}$. Each site contain replicated of data *x*. If the transaction of $T_{\alpha x,PM}$ get lock from site $i \in S(B_x)$ and other transaction will get lock from other site $j \in S(B_x) \mid j \neq i$. Each sites $i \in S(B_x)$ has its own lock manager (LM) that process a request for a lock from the transaction either the lock can be granted or not. If the lock is free, it is granted immediately otherwise, the lock manager will send a reject message and inserts the requesting transaction or node ID into the waiting list for the lock [7]. Each node is uniquely identified by its {site id:process id} pair and for the simplicity of explanation a unique number has assigned using integer numbers (0, 1, 2, 3, ..., n)to all transaction or node. The transactions or nodes will create elements of probe message (*InitID, VictimID, DepCnt, and RouteString)*.

3. Implementation

In this section, we present the implementation of the system. The purpose of this implementation is to illustrate that our system can detect and resolve deadlock problems.

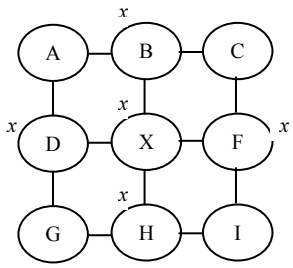

Fig. 2: A cluster with sixteen replication server.

In implementation phase, based on the NRG model [10] we use a cluster with nine replication server that are logically connected to each other in the form of two-dimensional 3 x 3 grid structure shows in Fig. 2.

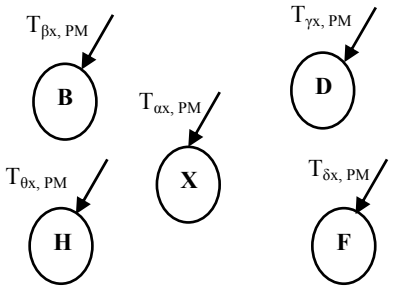

Fig. 3: Different set of transaction request a different site.

Data *x* in site X will be replicate to each site that adjacent with site X, which is site B, D, F and H. Without lost of generality, five different set of transaction T_α, T_β, T_γ, T_δ, *and* T_θ come to update data x at replica B, D, X, F, and H in the absence of system failures shows in Fig. 3.

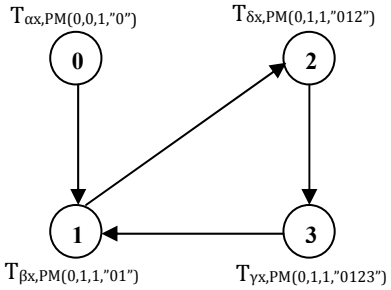

Fig. 4: Different set of transactions wait for each other.

The Fig. 4 shows that each transaction waiting for each other to obtain the lock. Without lost of generality assume that each transaction as a node that brings the probe message. Each node of transaction has its own node ID (0, 1, 2, 3) that shown in Fig. 4. Node 0, $T_{\alpha x,PM(0,0,1,"0")}$ has initiated the lock that waiting for another node, node 1. Node 1, $T_{\beta x,PM(0,1,2,"01")}$ is waiting for node 2, node 2, $T_{\delta x,PM(0,1,2,"012")}$ is waiting for node 3 and node 3, $T_{\gamma x,PM(0,1,2,"0123")}$ is waiting for node 1. In forwarding the probe message to other nodes, a node must check the emptiness of its *ProbeS* first. If it is found to be empty (*i.e.*,ill now no *probe* message is forwarded by this node), then it compares its own *DepCnt* value with probe's *DepCnt* value [7]. If this node's *DepCnt* is higher, then probe's *VictimID* and *DepCnt* values are updated with this node's ID and *DepCnt* values respectively; otherwise the values are kept intact. Before forwarding the *probe* message to all successors (the node that it's waiting) of this node, probe's *RouteString* field is updated by appending this node's ID at last of existing string (*i.e.*, concatenate operation). One copy of updated *probe* message is saved in *ProbeS* of this node. In Fig. 4 node 0 has initiated execution and send *probe* message (0,0,1,"0") to its successor node 1. As node 1's *ProbeS* is empty and *DepCnt* value is 1, the probe message is not update and store *probe* message as (0,1,1,"01") in *ProbeS* and forwarded to its successors or node 2. Nodes 2 and 3 have updated only the *RouteS* field of the *probe* message and forwarded to their successors.

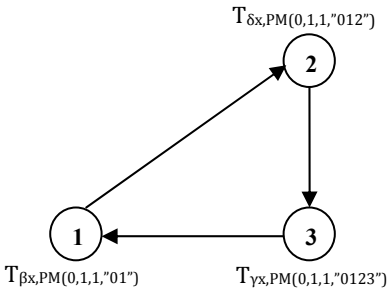

Fig. 5: Cycle of Deadlock

Deadlock is detected when *RouteString* of node 3, $T_{\gamma x,PM(0,1,2,"0123")}$ prefix with node 1, $T_{\beta x,PM(0,1,2,"01")}$ that start with "01" as shown in Fig. 5. When the deadlock is detected and the probe message will discard by the node that has detected a deadlock. Node 1 in Fig. 5 got back its forwarded *probe* and detected one deadlock cycles {1, 2, 3,1}. Deadlock will be resolve by aborting at least one node that involve in deadlock. In [7] selects the node with highest *DepCnt* value as victim and the deadlock detector node sends a *victim* message to all successors. If the detector node is not the initiator, it also sends the *victim* message to all *simply blocked* (node that is blocked but not a member of deadlock cycle) nodes [7]. On reception of this message, the *victim* node first forwards it to all of its successors and then releases all locks held by it and killed itself, other nodes delete deadlock detection information from their *ProbeS* memories [7]. Node 1 has detected as a victim because it has the highest *DepCnt* value amongst the members in any of the cycles. And to resolve the deadlock detection, node 1 as a victim kill itself or abort the lock and released it to another nodes. Node 1 is not the initiator, so it has also sent the *victim* message to *simply blocked* node 0. Nodes 3 stop further propagation of *victim* message. The Table 2 shows the experiment result that the NRGDD can detect and resolve the deadlock problems.

Table 2: The Experiment Result

Replica Time	X	B	H	D
t1	unlock(x)	unlock(x)	unlock(x)	unlock(x)
t2	Begin_transaction	Begin_transaction	Begin_transaction	Begin_transaction
t3	Write lock(x), counter _w =1	Write lock(x), counter _w =1	Write lock(x), counter _w =1	Write lock(x), counter _w =1
t4	wait	Wait	Wait	wait
t5	$T_{\alpha x,PM(0,0,1,"0")}$ Propagate lock: B	$T_{\beta x,PM(0,1,1,"01")}$ Propagate lock: I	$T_{\delta x,PM(0,1,1,"012")}$ Propagate lock: D	$T_{\gamma x,PM(0,1,1,"0123")}$ Propagate lock: B
t6	wait	wait	wait	wait
t7				Detect deadlock which is RouteString prefix with $T_{\beta x,PM(0,1,2,"01")}$, send victim message: $T_{\beta x,PM(0,1,2,"01")}$
t8		Receive Victim message		
t9		Propagate victim message: X, D		Stop propagate victim message
t10	Receive victim message		Receive victim message	Receive victim message
t11		abort or kill: $T_{\beta x,PM(0,1,2,"01")}$		
t12	Released	Released: B	Wait to lock D	Release: D

4. Conclusion

Managing transaction in distributed databases is important in order to ensure the transaction can occur properly. This paper presents a new algorithm to manage transaction on neighbour replication in distributed database system. Deadlock will occur on the transaction which is different set of transactions may required the same resources that obtain by another transaction. NRGDD has resolve the deadlock problem by sending the minimum number of probe message to detect the deadlock and it can resolve the deadlock to ensure the transaction can be done smoothly.

5. Acknowledgement

This work has been supported by the University Malaysia Pahang under grant RDU090306.

6. References

[1] Oracle, Release 2(9/2). Oracle9i Heterogeneous Connectivity Administrator's Guide. Oracle Copyright © 2001, 2002 Oracle Corporation.

[2] Shashi Bhushan, R. B. Patel and Mayank Dave. A Secure Time-Stamp Based Concurrency Control Protocol for Distributed Databases. *Journal of Computer Science*. Vol. 3, No.7, pp. 561-565, 2007.

[3] Gao, M. Dahlin, A. Nayate, J. Zheng and A. Iyengar. Improving Availability and Performance with Application-Specific Data Replication. *IEEE Trans. Knowledge and Data Engineering.* Vol. 17, No. 1, pp. 106-200, 2005.

[4] A.Noraziah, M. Mat Deris, Man, R.Norhayati, M.Rabiei, W.N.W. Shuhadah. Managing Transaction on Grid-Neighbour Replication in Distributed System. *International Journal of Computer Mathematics.* Vol. 86, No. 9, pp. 1-10, Sept 2008, DOI:10.1080/00207160801965198.

[5] B.M. Monjurul Alom, Frans Henskens, Michael Hannaford. Optimization Of Detected Deadlock View of Distributed Database. *2010 International Conference on Data Storage and Data Engineering*, 978-0-7695-3958-4/10 $26.00 © 2010 IEEE, DOI 10.1109/DSDE.2010.41.

[6] M. Tang, B. S. Lee, X. Tang and C. K. Yeo. The impact on data replication on Job Scheduling Performance in the Data Grid. *International Journal of Future Generation of Computer Systems*. Elsevier (22), pp. 254-268, 2006.

[7] Md. Abdur Razzaque, Md. Mamun-Or-Rashid, Choong Seon Hong. MC2DR: Multi cycle Deadlock Detection and Recovery Algorithm for Distributed Systems. R. Perrot et al. (Eds.): HPCC 2007, LNCS 4728, pp. 554-565, 2007 © Springer- Verlag Berlin Heidelberg 2007.

[8] B.M. Monjurul Alom, Frans Henskens, Michael Hannaford. Deadlock Detection Views of Distributed Database. *2009 Sixth International Conference on Information Technology: New Generations.* 978-0-7695-3596-8/09 $25.00 © 2009 Crown Copyright, DOI 10.1109/ITNG.2009.220.

[9] Aida. O. Abd El-Gwad, Ahmed. I. Saleh, Mai. M. Abd-ElRazik. A Novel Scheduling Strategy for an Efficient Deadlock Detection. 978-1-4244-5844-8/09/$26.00 © 2009 IEEE.

[10] Noraziah Ahmad. Neighbour Replica Failure Semantic Using NRTM in Distributed Environment. *International Conference on SE & SCS 2009, Malaysia.* Vol. 1, No. 1, pp. 530-533, Oct 2009.