Journal of Computer Science 8 (2): 265-271, 2012
ISSN 1549-3636
© 2012 Science Publications

Solving Two Deadlock Cycles through Neighbor Replication on Grid Deadlock Detection Model

[1]Noriyani Mohd Zin, [2]A. Noraziah, [1]Ahmed N. Abdalla, [2]Ainul Azila CheFauzi
[1]Faculty of Computer Systems and Software Engineering,
University Malaysia Pahang, 26300 Kuantan, Malaysia
[2]Faculty of Electric and Electronic, University Malaysia
Pahang, 26300 Kuantan, Pahang, Malaysia

Abstract: A data grid is compose of hundreds of geographically distributed computers and storage resources usually locate under different places and enables users to share data and other resources. **Problem statement:** Data replication is one of the mechanisms in managing data grid architecture that receive particular attention since it can provide efficient access to data, fault tolerance, reduce access latency and also can enhance the performance of the system. However, during transaction deadlock may occur that can reduce the throughput by minimizing the available resources, so it becomes an important resource management problem in distributed systems. **Approach:** The Neighbor Replication on Grid Deadlock Detection (NRGDD) transaction model has been developed to handle two deadlock cycle problems on grid. By deploying this method, the transactions communicate with each other by passing the probe messages. The victim message has been used to detect the deadlock when the number of waiting resource by other transaction is highest and become as the cause of deadlock occurs. In addition, this transaction must be aborted to solve the problem. **Results:** NRGDD transaction model are able to detect and solve more than one cycle of deadlocks. **Conclusion:** NRGDD has resolve the deadlock problem by sending the minimum number of probes message to detect the deadlock and it can resolve the deadlock to ensure the transaction can be done smoothly.

Key words: Replication, Distributed system, NRGDD, Probe message, Deadlock cycles

INTRODUCTION

Data grid (Muruganantham *et al.*, 2010) is the solutions that enable especially researcher to make their research on their specific fields. The researcher can know the latest issues that has been done by other researcher according to their fields of research such as method, technology, application and etc that has been used in enhancing the research fields. The concept of the computing grid arose from the need to share computing power, mostly for the jobs that use read only data sets as input (output from scientific experiments) (Noraziah *et al.*, 2009; Radi *et al.*, 2008). Consequently, the primary design of data management tools for grid computing was used to manage read-only data sets. A data grid is composed of hundreds of geographically distributed computers and storage resources usually located under different places and enables users to share data and other resources. The required for data grids because of the data is being produced at a tremendous rate and volume especially from scientific experiments in the fields of high-energy physics, molecular docking, computer micro-tomography and many others (Ahmad *et al.*, 2010a; 2010b). The grid computing requirements are more complex than distributed computing even though it is quite similar to normal distributed computing. Distributed computing refers to managing hundreds or thousands of computer systems that are individually more limited in their memory and processing power (Sashi and Thanamani, 2010). However, grid computing concentrates on the efficient use of a pool of heterogeneous systems with optimal workload management.

The major problem on grid environment is data management. In grid computing, there is no limitation on the number of users, departments or organizations. Besides that, the size of the data managed by data grids is continuously growing (Perez *et al.*, 2010). In Data Grid, when a user requests a data, a large amount of bandwidth could be spent to send the data from the server to the client. Furthermore, the delay involved could be high (Bsoul *et al.*, 2010). Data grid not only deals with efficient management but it is also deals with the placement and replication of large amounts of data.

Data replication (Sleit *et al.*, 2007; Noraziah *et al.*, 2007) is one of the technique or key components in data grid to increase availability and reliability of the data. Besides that, replication method can increase the

Corresponding Author: Noriyani Mohd Zin, Faculty of Computer Systems and Software Engineering, University Malaysia Pahang, 26300 Kuantan, Malaysia

system scalability, performance and fault tolerance (Fauzi *et al*., 2011; Mohammed, 2007). To speed up data access for data grid systems, data can be replicated in multiple locations, so that a user can access the data from nearby locations (Sashi and Thanamani, 2010). Furthermore, replication can reduce access latency; improve data locality, increase robustness, scalability and performance for distributed applications (Radi *et al*., 2008). Organizations need to provide current data to users who may be geographically remote and request distributed data around multiple sites in data grid (Ahmad *et al*., 2010a). A data grid is composed of hundreds of geographically distributed computers and storage resources usually located under different places and enables users to share data and other resources. Replication strategies determine when and where to create a replica, taking into account of the factors such as request number of the data, network conditions, storage availability of nodes, etc (Perez *et al*., 2010; Sun *et al*., 2009).

Read-One-Write-All (ROWA), Branch Replication Scheme (BRS), Hierarchical Replication Scheme (HRS) and Neighbor Replication on Grid (NRG) are the example of existing replication techniques. In ROWA technique (Noraziah *et al*., 2010) read operation has low communication cost. Meanwhile, this technique restricts the availability of write operations since they cannot be executed at the failure of any copy. In BRS technique (Perez *et al*., 2010), the clients that who request for the data file, the replicas are created as close as possible to them. The root replica grows toward the clients in a branching way, slip replicas into several sub replicas (Ahmad *et al*., 2010c; 2010d). In this technique, the replica tree will be growing based on the client needs. In HRS technique, a hierarchical replication consists of a root database server and one or more database servers organized into a hierarchy topology (Perez *et al*., 2010). Using this technique, the data will be replicated or copy at all sites and has the highest storage of use. Neighbor Replication on Grid (NRG) considers only neighbors obtain a data copy where the neighbors are assigned with binary vote one and zero otherwise (Ahmad *et al*., 2010c; 2010d). NRG requires significantly lower communication cost for an operation, while providing high system availability, due to the minimum number of quorum size required executing the transaction (Noraziah *et al*., 2009).

In replication, the concurrency control and deadlock (Senouci *et al*., 2007, Mohammed, 2007) handling is the most important problem that must have manages when sharing any data in distributed systems. 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 can reduce the throughput by minimizing the available resources, so it becomes an important resource management problem in distributed systems (Srinivasan and Rajaram, 2011). There are two major deficiencies in deadlock where any other process cannot grant all the resources that held by the deadlocked processes and the deadlock persistence time will added to the reaction time of each process involved in the deadlock.

In this study, without lost of generality the terms nodes is used to indicate as transaction for the explanation. The new model namely Neighbor Replication Grid Deadlock Detection (NRDGG) (Zin *et al*., 2011b) is proposed to detect and solve the deadlock. NRGDD model is formed by combining the Multi-Cycle Deadlock Detection and Recovery (MC2DR) algorithm (Razzaque *et al*., 2007) and Neighbor Replication on Grid (NRG) replication model (Noraziah *et al*., 2009). NRG has been proposed in our previous work. NRG able to maximize the write availability with low communication cost due to the minimum number of quorum size required. However, the study not discuss on how to resolve the deadlock detection. The purpose of this research is to show how the new algorithm can detect the existence of real deadlock and resolve it through the NRG replication model.

MATERIALS AND METHODS

Deadlock occurs when different set of transaction waiting for each other to obtain the same resource, thus the transaction become stuck. This part describes NRG Deadlock Detection (NRGDD) transaction model which involves T as a transaction, 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 $T_{\theta.}$ *Consider* $\lambda = \alpha, \beta, \gamma, \delta, \theta$ where it represent different group for the transaction T. Meanwhile, PM is a probe message. It contain a set of probe messages where PM = {*InitID*, *VictimID*, *DepCnt*, *RouteString*}. Table 1 shows the probe message details description.

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.

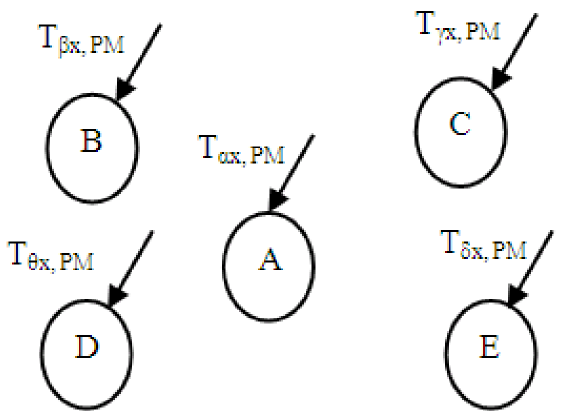

Fig. 1: Different set of transaction request a different site

Define the following probe message:

a) NRG transaction elements T_α = {$T_{\alpha x,PM}$ | *PM=InitID, VictimID, DepCnt, RouteString*} where $T_{\alpha x,PM}$is a probe message elements of $T_\alpha$transaction.
b) NRG transaction elements T_β = {$T_{\beta x,PM}$ | *PM=InitID, VictimID, DepCnt, RouteString*} where $T_{\beta x,PM}$is a probe message elements of $T_\beta$transaction.
c) NRG transaction elements T_γ = {$T_{\gamma x,PM}$ | *PM=InitID, VictimID, DepCnt, RouteString*} where $T_{\gamma x,PM}$is a probe message elements of $T_\gamma$transaction.
d) NRG transaction elements T_δ = {$T_{\delta x,PM}$ | *PM=InitID, VictimID, DepCnt, RouteString*} where $T_{\delta x,PM}$is a probe message elements of $T_\delta$transaction.
e) NRG transaction elements T_θ = {$T_{\theta x,PM}$ | *PM=InitID, VictimID, DepCnt, RouteString*} where $T_{\theta x,PM}$is a probe message elements of $T_\theta$transaction.

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*. Meanwhile, 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*.

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 $S(B_x)$ = {A,B,C,D,E}. Each site contain replicated of data *x*. If the transaction of $T_{\alpha x,PM}$ get lock from site i Є $S(B_x)$ and other transaction will get lock from other site j Є $S(B_x)$ | j ≠ i.

Each sites i Є $S(B_x)$ has its own Lock Manager (LM) that process a request for a lock from the transaction either the lock can be granted or not. The lock is granted immediately when it is free otherwise; the lock manager will send a reject message to the requesting transaction or node ID, then inserts it into the waiting list for the lock. 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*), Tλ = {T λx, PM | PM=*InitID*, *VictimID*, *DepCnt*, *RouteString*}.

Implementation: In this phase, we present the implementation of the system. The purpose of this implementation is to illustrate that our system can detect and resolve deadlock problems. This phase will detect two existing cycle of deadlock, for the previous research one cycle of deadlock has been detected (Zin *et al*., 2011a). In implementation phase, based on the NRG model we use a cluster with nine replication servers that are logically connected to each other in the form of two-dimensional 3×3 grid structure. Data object *e* in this experiment represent the data object *x* in NRGDD Transaction Model. Data *e* in site E will be replicate to each site that adjacent with site E, 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 *e* at replica B, D, E, F and H in the absence of system failures shows in Fig. 2.

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, 4). Figure 3 shows that each transaction waiting for each other to obtain the lock. Node 0, $T_{\alpha e,PM(0,0,1,"0")}$ has initiated the lock that waiting for another node, node 1. Node 1, $T_{\beta e,PM(0,1,2,"01")}$ is waiting for node 2 and 3, node 3,$T_{\gamma e,PM(0,1,2,"013")}$ is waiting for node 2 and node 2,$T_{\delta e,PM(0,1,2,"012")}$ is waiting for node 4,$T_{\theta e,PM(0,1,2,"0124")}$ where it's waiting for node 1.

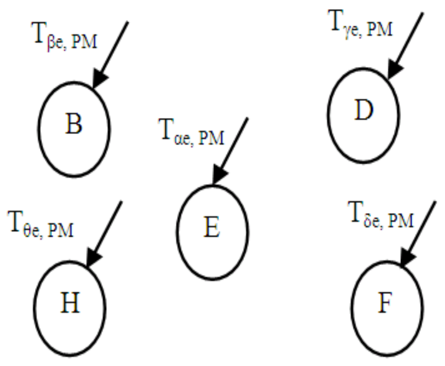

Fig. 2: Different set of transaction request to update data e at different sites

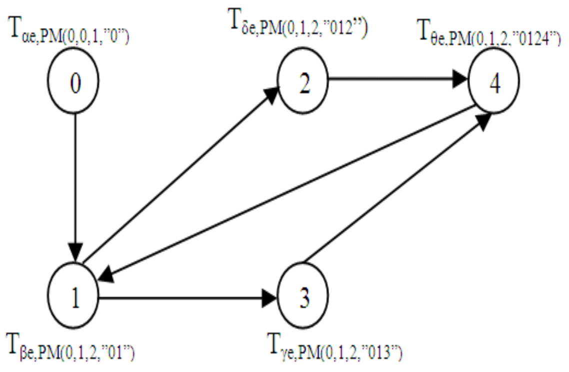
$T_{\alpha e,PM(0,0,1,"0")}$
$T_{\delta e,PM(0,1,2,"012")}$
$T_{\theta e,PM(0,1,2,"0124")}$
0
2
4
1
3
$T_{\beta e,PM(0,1,2,"01")}$
$T_{\gamma e,PM(0,1,2,"013")}$

Fig. 3: Different set of transactions wait for each other to update data e

In forwarding the probe message to other nodes, a node must check the emptiness of its *ProbeS* first. It will compare its own *DepCnt* value with probe's *DepCnt* value when its *ProbeS* is empty. 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. For example, in Fig. 3 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 2, it has updated the *probe* message, stored the modified *probe* (0,1,2,"01") in *ProbeS* and forwarded to its successors 2 and 3. Nodes 2, 3 and 4 have updated only the *RouteS* field of the *probe* message and forwarded to their successors.

Deadlock is detected when *RouteString* of node 4, $T_{\theta e,PM(0,1,2,"0124")}$ prefix with node 1, $T_{\beta e,PM(0,1,2,"01")}$ that start with "01". The probe message discards by the node that has detected a deadlock. Deadlock cycle can be detected at any node when the travelled path of *probe* message makes a dependency cycle.

Deadlock is resolved by aborting at least one node that involves in deadlock, hence other node can get the lock that has been released. In resolving this, the node with highest *DepCnt* value has selected as the victim and its will 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. On receiving of this message, the victim node first send it to all of its successors or the resources that is waiting for and then releases all locks held by it and killed itself, other nodes delete deadlock detection message from their *ProbeS* memories.

In Fig. 4 shows one deadlock cycle have detected. Node 1 got back its forwarded probe and detected one deadlock cycles {1, 2, 4, 1}.Meanwhile, Fig. 5 shows the second deadlock cycle has detected. Node 1 got back its forwarded probe and detected one deadlock cycles {1, 3, 4, 1}.

Based on the Fig. 4 and Fig. 5, there are two cycles of deadlock that send probe message to node 1.

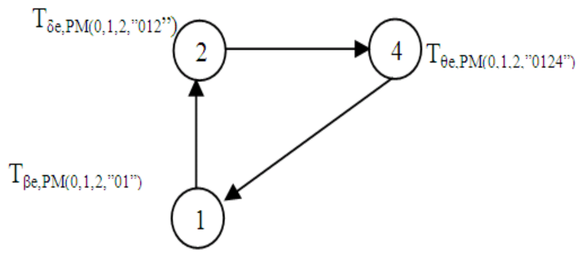
$T_{\delta e,PM(0,1,2,"012")}$
2
4
$T_{\theta e,PM(0,1,2,"0124")}$
$T_{\beta e,PM(0,1,2,"01")}$
1

Fig. 4: First cycle of the deadlock

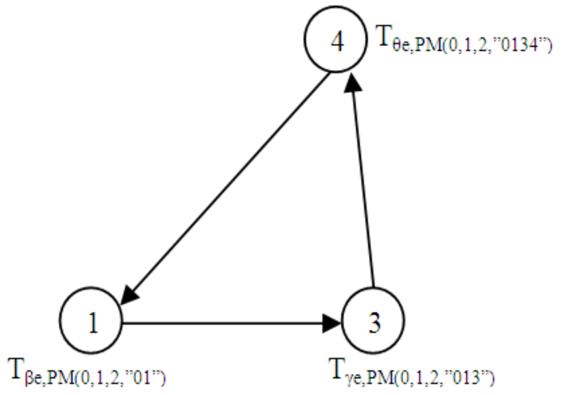
4
$T_{\theta e,PM(0,1,2,"0134")}$
1
3
$T_{\beta e,PM(0,1,2,"01")}$
$T_{\gamma e,PM(0,1,2,"013")}$

Fig. 5: Second cycle of the deadlock

However, only one cycle will be detected as deadlock cycle, if the probe message of deadlock cycle {1,2,4,1} is receive first then from the node 3 is discarded or vice versa. So, consider that Fig. 4 is the cycle of deadlock that must be solve when the cycle in Fig. 5 has been discarded. 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 killed itself or aborts 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 4 stop further propagation of victim message.

RESULT AND DISCUSSION

In implementation, there are two cycles of deadlock has been detected. However, only one cycle will be solved and other cycle will discard. In the experiment will consider that cycle in Fig. 4 is first receiving probe message than Fig. 5. The Table 2 shows the result of the two cycle deadlock that has been detected.

Table 2: Result for two deadlock cycles detection

Replica Time	B	D	F	H
t1	unlock(e)	unlock(e)	unlock(e)	unlock(e)
t2	Begin_tran saction	Begin_tran saction	Begin_tran saction	Begin_tran saction
t3	Write lock(e), counter _w =1	Write lock(e), counter _w =1	Write lock(x), counter _w =1	Write lock(x), counter _w =1
t4	Wait	Wait	Wait	wait
t5	$T_{\beta e,PM(0,1,2,"01")}$ Propagate lock: F	$T_{\delta e,PM(0,1,2,"012")}$	$T_{\gamma x,PM(0,1,2,"013")}$ lock:H	$T_{\theta x,PM(0,1,2,"0124")}$ Propagate lock: F
t6	Propagate lock: D		Wait	Wait
t7	Wait	Propagate lock:H		
t8				Propagate lock: B
t9	Receive request from H			Receive request from D and propagate lock: B
t10	Receive 2nd request from H			

Table 3: Result for deadlock cycle resolution

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

In Table 2 present that Fig. 4 will be solve and Fig. 5 will be discarded. At t9 the replica B receives requested from replica H first after F requested for H then at t10 replica B receive request from H after D requested for H. When cycle of the deadlock has been detected, the resolve strategy will be applied. Table 3 shows the results to resolve the deadlock cycle. At t5, node 4 from replica H send probe message to B. Detect deadlock is detected when RouteString from $T_{\gamma e,PM(0,\ 1,\ 1,\ "0123")}$ is prefix with $T_{\beta e},_{PM\ (0,\ 1,\ 2,\ "01")}$, which is “01”. Replica B receive victim message from replica H and it will propagate victim message to replica E and F and replica F will stop propagate the victim message to other replica. Then at time t11 the node 1, $T_{\beta e,PM(0,\ 1,\ 2,\ "01")}$ will be abort and killed. The lock at replica B will be released at t12. Other node of transactions can request a lock from replica B. Finally, the NRGDD can resolve deadlock cycle after the cycle of deadlock has detected.

CONCLUSION

Managing transaction in distributed databases is important in order to ensure the transaction can occur properly. The novel contribution of this study is a mechanism to handle two deadlock cycles problem by using Neighbor Replication on Grid Deadlock Detection (NRGDD) transaction model. Normally, the deadlock occurs when transaction in different set of transactions request the same resources that obtain by another transaction. NRGDD has resolve the deadlock problem by sending the minimum number of probes message to detect the deadlock and it can resolve the deadlock to ensure the transaction can be done smoothly.

REFERENCES

Ahmad, N., A.A.C. Fauzi, R.M. Sidek, N.M. Zin and A.H. Beg, 2010a. Lowest Data Replication Storage of Binary Vote Assignment Data Grid. In: Networked Digital Technologies, Part II, Zavoral F., (Eds.). Springer, ISBN: 3642143059, pp: 466-473.

Ahmad, N., N.M. Zin, R. Mohd. S.M.F.J. Klaib and M.H.A. Wahab, 2010b. Neighbour Replica Transaction Failure Framework in Data Grid. In: Networked Digital Technologies, Part II, Zavoral F., (Eds.). Springer, ISBN: 3642143059, pp: 488-495.

Ahmad, N., M.F.J. Klaib and R.M. Sidek, 2010c. Failure semantic of neighbour replication grid transaction model. Proceedings of the 10th IEEE International Conference on Computer and Information Technology June 29-July 1, IEEE Xplore Press, Bradford, pp: 668-673. DOI: 10.1109/CIT.2010.132

Ahmad, N., A.N. Abdalla and R.M. Sidek, 2010d.Data replication using read-one-write-all monitoring synchronization transaction system in distributed environment. J. Comput. Sc. 6: 1033-1036.DOI:10.3844/jcssp.2010.1095.1098

Bsoul, M., A. Al-Khasawneh, E.E. Abdallah, Y. Kilani, 2010. Enhanced fast spread replication strategy for data grid. J. Network Comput. Appli. 34: 575-580. DOI: 10.1016/j.jnca.2010.12.006

Fauzi, A.A.C., A. Noraziah, N.M. Zain, A.H. Beg and N. Khan *et al*., 2011. Handling fragmented database replication through binary vote assignment grid quorum. J. Comput. Sci., 7: 1338-1342. DOI: 10.3844/jcssp.2011.1338.1342

Mohammed, T.S., 2007. Performance improvement and deadlock prevention for a distributed fault diagnosis algorithm. J. Comput. Sci. 3: 107-112.DOI:10.3844/jcssp.2007.107.112

Muruganantham, S., P.K.Srivastha and Khanaa, 2010. Object based middleware for grid computing. J. Comput. Sci. 6: 336-340. DOI:10.3844/jcssp.2010.336.340

Noraziah, A., M.M. Deris, M.Y.M. Saman, R. Norhayati, M. Rabiei and W.N.W. Shuhadah, 2009. Managing transactions on grid-neighbourreplication in distributed systems. Int. J. Comput. Math., 86: 1624-1633. DOI: 10.1080/00207160801965198

Noraziah, A., M.M. Deris, N.A. Ahmed, M.Y.M. Saman and R. Norhayati *et al*., 2007. Preserving data consistency through neighbor replication on grid daemon. Am. J. Applied Sci. 4: 751-758. DOI: 10.3844/ajassp.2007.751.758

Perez, J.M., F. García-Carballeira, J. Carretero, A. Calderón and J. Fernández, 2010. Branch replication scheme: A new model for data replication in large scale data grids. Future Generat. Comput. Syst., 26: 12-20. DOI: 10.1016/j.future.2009.05.015

Radi, M., A. Mamat, M.M. Deris, H. Ibrahim and S. Shamala, 2008. Access weight replica consistency protocol for large scale data grid. J. Comput. Sci. 4: 103-110.DOI:10.3844/jcssp.2008.103.110

Razzaque, M.A., M. Mamun-Or-Rashid and C.S. Hong, 2007. MC2DR: Multi-cycle deadlock detection and recovery algorithm for distributed systems. High Performance Comput. Commun., 4782: 554-565.DOI: 10.1007/978-3-540-75444-2_53

Sashi, K. and A.S. Thanamani, 2010. Dynamic replication in a data grid using a Modified BHR Region Based Algorithm. Future Generat. Comput. Syst. 27: 202-210. DOI: 10.1016/j.future.2010.08.011

Senouci, M., A. Liazid and D. Benhamamouch, 2007. Towards an exclusion mutual tolerant algorithm to failures. J. Comput. Sci. 3: 43-46.DOI: 10.3844/jcssp.2007.43.46

Sleit, A., W. AlMobaideen, S. Al-Areqi, A. Yahya, 2007. A dynamic object fragmentation and replication algorithm in distributed database systems. Am. J. Applied Sci. 4: 613-618.DOI:10.3844/ajassp.2007.613.618

Srinivasan, S. and R. Rajaram, 2011. A decentralized deadlock detection and resolution algorithm for generalized model in distributed systems. Distributed Parallel Databases, 29: 261-276. DOI 10.1007/s10619-011-7078-7

Sun, X., J. Zheng, Q. Liu and Y. Liu, 2009. Dynamic data replication based on access cost in distributed systems. Proceeding of the 4th International Conference on Computer Sciences and Convergence Information Technology, Nov. 24-26, IEEE Xplore Press, Seoul, pp: 829-834. DOI:10.1109/ICCIT.2009.198

Zin, N.M., A. Noraziah and A.A.C. Fauzi, 2011b.Neighbour replication on grid deadlock detection framework. Digital Enterprise Inform. Syst., 194: 350-357. DOI: 10.1007/978-3-642-22603-8_32

Zin, N.M., A. Noraziah, A.H. Beg, A.A.C. Fauzi, 2011a. Deadlock Detection and Resolution in Neighbour Replication on Grid. Proceedings of the International Conference on Computer Communication and Management, (CCM’2011), IACST Press, Singapore, pp: 426-430.