Update-Ordering For Database Consistency in Peer-to-Peer Environments

W. N. Shuhadah[1], M.Mat Deris[2], A. Noraziah[1], M.Y.Saman[1], M.Rabiee[1]

[1]University College of Science & Technology Malaysia, Faculty of Science and Technology, 21030, Mengabang Telipot, Kuala Terengganu, Malaysia.
E-mail: shud_81@yahoo.com

[2]University College of Technology Tun Hussein Onn, Faculty of Information & Technology Multimedia,
86400 Parit Raja, Batu Pahat, Johor, Malaysia.
E-mail: mmustafa@kuittho.edu.my

Abstract

Database consistency is one of the major issues in replicated database in peer to peer environment. The logical design for the replicated nodes and the transaction management mechanism are two aspects that give a serious impact to the performance and the consistency of replicated database. This paper proposes a new model that combines the Neighbor Replication on Grid (NRG), where the data is replicated to the neighbors of the grid with the Update Ordering approach. The performance comparison shows that the proposed mechanism is greatly improve the performance of the replicated database in peer to peer environment up to two orders of magnitude while preserving the data consistency.

1. Introduction

Peer-to-peer (P2P) environment is undoubtedly one of the most touted topics in the internet. As new communication technologies are emerging, P2P concepts become reality and allow for even higher degrees of flexibility in distributed databases [3, 4]. Replication is a useful technique for a distributed database systems where an object will be accessed from multiple locations such as from a local area network environment or geographically distributed world wide [2]. Replication supports a variety of applications that have very different requirements. Some applications are adequately supported with only limited synchronization between the copies of the database and the corporate database system, while other applications demand continuous synchronization between all copies of the database [5]. Most of all, replication jeopardizes data consistency. In turn, mechanisms have to be employed to enforce the data consistency. Maintaining the data consistency is very expensive [7]. A common practice is then to relax the data consistency as much as possible to give rise to better system performance.

The existing replication control mechanism can be categorized into two spectrums: the logical design for the replicated nodes and its transaction management mechanism. For the logical design point of view, the protocol focuses on the number of copies being updated upon write operation. The examples include read-one-write-all (ROWA) [8], and the quorum techniques where one of which is Neighbor Replication on Grid (NRG) [7]. The NRG imposes the intersection requirement between read and write operations. This technique produces high availability for update-frequent operations by imposing a neighbor binary vote assignment to the logical grid structure on data copies. Also, it reduces the waiting time by decreasing the number of copies being contacted upon executing the write operation.

For a transaction management mechanism point of view, the protocol determines how to manage the transaction (read and write) on the replicated data in order to preserve the data consistency. Various existing transaction management protocol is developed for a transactional model. The examples include the model proposed in [1], [7], [8]**.** Two-phase commit [5] protocol is the most common approach to providing a consistent view for a transactional model in a distributed database system. However, data replication developed for transactional models are very strict since

1-4244-0674-9/06/$20.00 ©2006 IEEE.

one-copy-serializability is often required in order to maintain the ACID (atomicity, consistency, isolation and durability) property. Therefore, a long response time may occur and a low system throughput rate results.

Not all replication systems require such a strong transactional semantics. Thus, the update ordering protocol has been proposed in [6]. Update ordering is an alternative data consistency model with weaker semantics with those of one-copy-serializability since the model let replicas execute the same set of update requests in a sensible order. This approach can be applied in many distributed applications with less strict consistency requirements such as applications in retail and wholesale and applications in information storage and retrieval.

In this paper, without loss of generality, the terms node and site will be used interchangeably. The purpose of this paper is to combine and reconcile NRG logical design and update ordering approach to improve the performance of replicated systems in terms of response time while still preserve the data consistency.

This paper is organized as follows: In Section 2, we review the NRG logical design and update ordering approaches which are then compared with the protocol proposed in [1]. In Section 3, the reconciliation model is presented. Section 4, the simulation and the example for the potential scenario of the model is given. The performance evaluation of the proposed model is in terms of systems response time is presented in Section 5 while the conclusion is presented in Section 6.

2. Model

2.1 Replica Control Technique

A distributed system with replicated servers consists of many sites interconnected by a communication network. In NRG, all sites are logically organized in the form of a two-dimensional grid structure. For example, if an NRG consists of twenty-five sites, it will be logically organized in the form of 5 x 5 grid as shown in Figure 1. We use R to denote the set of all sites in a replicated system:

$R = \{ R_1, R_2 \ldots, R_n \}$, where n = total number of sites in replicated system.

Each site has master data item. Let O be the set of all data item that can be reached by the update request for replicated system. Thus,

$O = \{ d_1, d_2, \ldots, d_i, \ldots..., d_n \}$, where $i = 1,2,3,..,n$

and d_i is a master data item for site R_1.

A site is either operational or failed and the state of (operational or failed) of each site is statistically independent to the others. When a site is operational, the copy at the site is available; otherwise it is unavailable.

Definition 1: A site X is a neighbor to site Y, if X is logically-located adjacent to Y.

A data will replicate to the neighboring site from its primary site. The number of data replication, $r \leq 5$.

For example, from Figure1, data from site 1 will replicate to site 2 and site 4 which are its neighbors. Site 5 has four neighbors, which are sites 2, 4, 6, and 8. As such, site 5 has five replicas. For simplicity, the primary site of any data file and its neighbors are assigned with vote one and vote zero otherwise. This vote assignment is called binary vote assignment on grid. A neighbor binary vote grid assignment on grid, G, is a function such that $B(R_i) \in \{0,1\}$, $1 \leq i \leq n$, where $G(R_1)$ is the vote assign to site R_i. This assignment is treated as an allocation of replicated copies and a vote assigned to the site results in a copy allocated at the neighbor. That is, 1 vote $\equiv$ 1 copy.

Let $S(B)$ be the set of sites at which replicated copies of data items are stored corresponding to the assignment B. Then

$S(B) = \{R_1 | B(R_1) = 1, 1 \leq i \leq n\}$.

For any site R_i, where $R_i \in S(B_{d_x})$, R_i is said as a *replica* for data item d_x. Therefore, for an

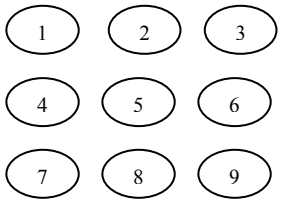

Figure 1: A grid organization of 9 copies of an object

r-replica group, we use C to denote a number of replicas for a particular data item d_x.

$C = \{ R_1^t, \ldots, R_i^s, \ldots, R_n^r \}$, where $t,s = 1,2,\ldots r$, $t \neq s$, $i = 1,2,\ldots,n$, $R_i^t \in S(B_{d_x})$ and $r \leq 5$.

2.2 Update Ordering

An update ordering approach is a model which using a set of ordering constraints to express the corresponding set of operations provided by a replica group. The ordering implementation takes account of detailed inter-operation semantics denoted by commutative operations and causal operations to

reduce unnecessary delay. Each replica will execute the same set of update operations in a sensible order which is confined to the set of ordering constraints but maybe different at replicas. When an update requests are propagated to a group of replicas by different replicas concurrently, their arriving orders at replicas maybe different. Figure 2 depicts this scenario by considering R_i where $R_i \in S(B_{d_1})$ with referring grid organization in Figure 1. Four operations u_1,u_2,u_3 and u_4 are send to a three-replica group { R_1, R_2, R_6}, they arrive at in R_1, R_2,and R_6 the order of (u_1,u_2,u_3, u_4), (u_2,u_3,u_4, u_1) and (u_1,u_3,u_2, u_4), respectively.

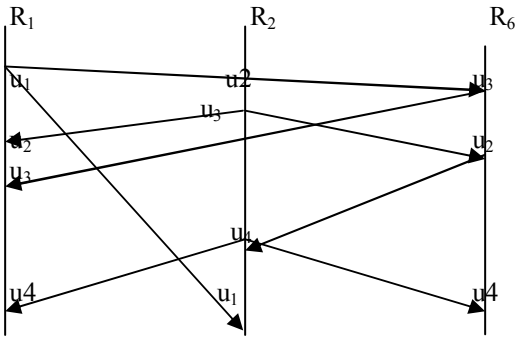

Figure 2: A scenario of message arriving orders with four operations u_1,u_2,u_3 and u_4 are send to a three-replica group { R_1, R_1, R_1}.

This scenario is the result of different network latencies on communication links between members on which the group of replicas are running. To ensure the correct semantics of the replicated service system, a sensible arriving order of update operations has to be defined and enforced over the whole replica group. In general, ordering constraints are categorized into four types: *FIFO, causal, total* and *total + causal* to reflect different semantic requirements of the replicated system and its client. *FIFO* and *causal* orderings are the ones often required from the client's point of view, whereas *total* ordering is often required from the replicas group's point of view. *Total + causal* is the integrated constraint to give the satisfaction to both parties: clients and the replica group.

Let U be a set of update request in the system.

Then,U={$u(d_x)$|u=update request, d_x= data item.}

An update request, $u(d_x)$ received by a replica R_i directly from its client is said to be *originated* from R_i. Thus, R_1 is also said as a *primary site* for update request $u(d_x)$. Any replica R_j, where $i \neq j$ who received an update request, $u(d_x)$ from other replica, R_j is said as *neighbor site* for a particular data item d_x. We also need to distinguish a *received* request from a *deliverable* request. When a request is *received* by a replica, it is stored in a buffer/log and awaits to be checked on its ordering constraint. Once its ordering constraint is satisfied, that request is *executable* or *deliverable*. In other words, that request is ready to be executed by the replica.

For any replica R_i, where $R_i \in S(B_{d_x})$, R_i will only allowed to receive only an update request for a particular data item d_x. We define $T_{S(B_{d_x})}$ as a set of update request that will be allowed to receive by a set of replica group $S(B_{d_x})$. Thus,

$$T_{S(B_{d_x})} = \{u_1(d_x), u_2(d_x), \ldots\ldots, u_k(d_x)\}$$

3. Simulation

All experiments are performed using two set of simulator representing two set of models; our reconciliation model and the existing ones which has been proposed in [1]. The simulator is written in C++ and has been used to simulate the update execution for both models over the same database environment. We do not consider the contribution of network delay in our simulation activities. Each replica in a particular replica group is assumed to receive an update request eventually after it is sent from the original replica. In this section, we only discussed the example for our reconciliation model since the existing ones has been discussed explicitly in [1].

The reconciliation model proposed in this paper can be applied in many distributed applications with less strict consistency requirements, such as applications in retail and wholesale and applications in information storage and retrieval. In this paper, we use a sales information system as an example in our simulation model to show the potential use of this model.

The following shows an example of potential scenario in our simulation model.

Example:

In this example, we consider 9 sites, (R_1, R_2,…R_9) which are logically organized in 3 x 3 grid structure and each site holds a master data item, (d_1,d_2,…,d_9) respectively. Supposed there are 10 update request, (u^1,u^2,u^3,….,u^{10}) which consists the combination of four update operations, u_1,u_2,u_3 and u_4 representing the operation of *addStock*, *deleteStock*, *sendMessage* and *replyMessage* respectively. Each update request reached at various site and their arriving patterns at each site are shown in Figure 5.

Each update operations can be categorized into three operation set, i.e: $Total_{op}$ = $\{u_1, u_2\}$, $Comm_{op}$ = $\{u_3, u_4\}$ and $Causal_{op}$ = $\{u_2, u_4\}$. Each update request carries the information for their own data items and it may be different between each update requests. Thus, the simulator is then need to identify a particular replica group for each update request by identifying their data item.

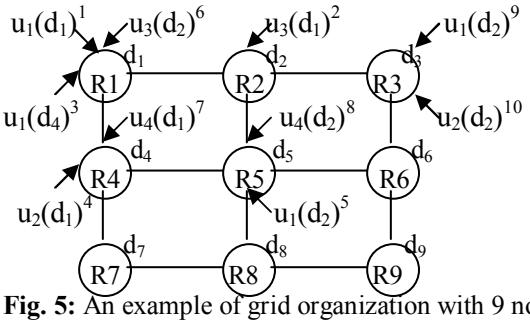

Fig. 5: An example of grid organization with 9 nodes

By analyzing all update request shown in Figure 2, the summary of replica groups for each update request are as below:

$T_{S(B_{d_1})}$ ={$u_1(d_1)^1$,$u_3(d_1)^2$,$u_2(d_1)^4$, $u_4(d_1)^7$}

$T_{S(B_{d_2})}$ = { $u_1(d_2)^5$,$u_3(d_2)^6$,$u_4(d_2)^8$, $u_1(d_2)^9$}

$T_{S(B_{d_4})}$ = { $u_1(d_4)^3$ }

All update requests in $Total_{op}$ in different replica group will be assigned with different set of counter.

Since $Comm_{op}$ = $\{u_3, u_4\}$, the execution for u^2, u^6, u^7 and u^8 will be based on the FIFO protocol, while for every update request in $Causal_{op}$, their execution will be based on the VT protocol. For u^4 and u^{10}, they carries a *time stamp* (TS) which consists of two fields. As a result, the execution orders for each update requests at their original replicas for u^1,u^2,u^3,u^4,u^5,u^6,u^7,u^8,u^9 and u^{10} are 1,1,1,2,1,1,2,2,2, and 3 respectively.

There is no prerequisite for u^1, u^3 and u^5 to be executable as they are the first operations issued in their own replica group. Thus, they can be executed concurrently at their original replica without being deferred for the arriving of any other update request while for u^2 and u^6, they also can be executed concurrently at their original replica since both of them are commutative to each other. For u^4, it carry TS and can only be executed after u^1 has been executed. u^7 and u^8 are both in $Causal_{op}$. Thus, they can only be executed concurrently at their original replicas after u^2 and u^6 has been executed respectively. For u^{10}, it carry TS and can only be executed after the execution of u^9 and it will be the last update request that will be executed at its original replicas for this example.

4. Results

The performance evaluation is based on response time over update request. In this paper, we compare the response time for our reconciliation model with the existing model which has been proposed by Baruch Awerbuch et al. in [1] with respect to update operations. All our experiments are carried out in the same database environment for both models.

The performance evaluation for total, total + commutative and total + causal operations is shown in figure 6. For total operations, all update requests are in $Total_{op}$ and they are conflicting to each other. For total + commutative operations, all update request are also in $Total_{op}$, but there are commutative pairs of update request that received by the sequencer. For each number of update request, we identify the maximum number of commutative pairs that possibly to have in each number of update request. For example, for 5 and 10 update request that received by the sequencer, the maximum number of commutative pairs for each number of update request are 2 and 5 pairs respectively. For total + causal operations, all update request are both in $Total_{op}$ and $Causal_{op}$. The execution of these three update operations bring out the same output for their response time since they are using the same protocol for their execution.

From Figure 6,7, and 8, the execution for commutative operations produced the lowest response time. The reason for this is that when receiving a commutative operation, the request can be handled right away at a replica. Whereas for total-ordering request, the request is sent to the sequencer to get the unique sequence number before it can be handled, which generates a long time delay compared to a commutative operations. The response time for the caused-by operation is slightly higher than the response

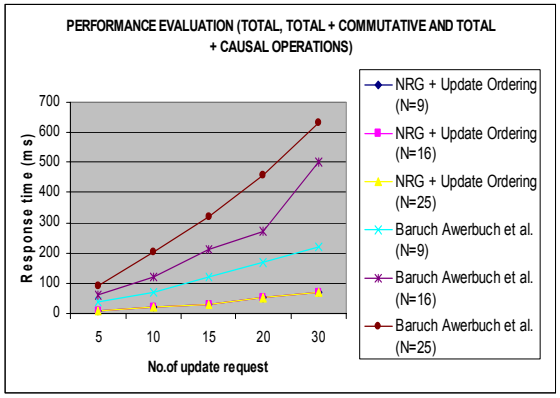

Figure 6: Performance evaluation for total, total + commutative, total + causal operations

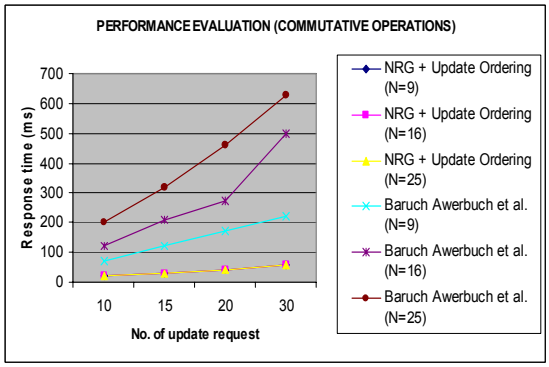

Figure 7: Performance evaluation for Commutative operations.

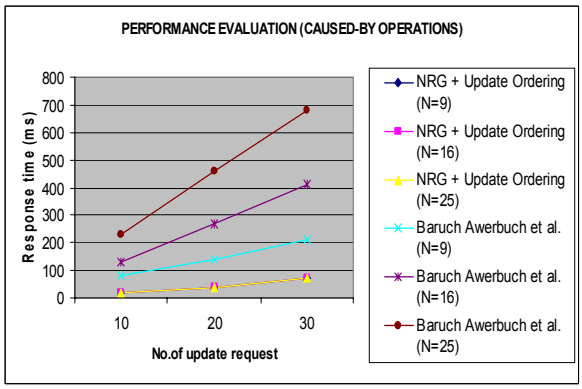

Figure 8: Performance evaluation for Caused-by operations.

time for commutative but it still lower than the execution of total-ordering operations. The caused-by operations is executable as long as its causal update request has been executed. Thus, a longer time delay is detected for its execution especially at a higher number of update requests.

5. Conclusions

A new reconciliation model has been proposed to maintain the database consistency in peer to peer environments. The NRG logical design has been reconcile with Update Ordering approach for this proposed model. We then analyzed the system performance in terms of an important response time for the execution of update request by the replicated system. The performance analysis shows the following findings: Firstly, an Update Ordering approach reduces the unnecessary delay and brings a better response time upon update request by allowing the definition of ordering constraint on each update operation, so further give a better concurrency rate to improve the systems performance. Secondly, the reconciliation by combining the NRG logical design and Update Ordering approach has greatly improved the performance for replicated systems up to two orders of magnitude while still maintain the replicated database consistency.

6. References

[1]B. Awerbuch, C. Tutu, Maintaining Database Consistency in Peer to Peer Networks, Technical Report, CNDS-2002-2, February 2002.

[2] M. D. Mustafa, B. Nathrah, M. H. Suzuri, M. T. Abu Osaman, Improving Data Availability Using Hybrid Replication Technique in Peer to Peer Environments, Journal of Interconnection Networks, vol. 5, no. 3, (2004) 299-312.

[3]J. Holiday, R. Steinke, D. Agrawal, and A. El Abbadi, Epidermic Algorithms for Replicated Databases, IEEE Trans. On Knowledge and Data Engineering, vol.15, 5(2003), pp. 1218-1238.

[4] O. Wolfson, S. Jajodia, and Y. Huang, An Adaptive Data Replication Algorithm, ACM Transactions on Database Systems, vol.22, 2(1997), pp.255-314.

[5] T. Connolly, C. Begg, Database Systems: A Practical Approach to Design, Implementation and Management, Edisson Wesley, 4th edition, 2005.

[6] W. Zhou, L. Wang, W. Jia, An Analysis of Update Ordering in Distributed replication Systems, Future Generation Computer Systems 20 (2004) 565-590.

[7] M. D. Mustafa, A. Noraziah, M. Y Saman, A. Noraida, Y. Yuan, High System Availability Using Neighbor Replication on Grid, IEICE Transactions on Information and Systems, vol. E87-D, no. 7, July 2004.

[8]J. Holliday, D. Agrawal, A. E. Abbadi, Database Replication Using Epidemic Communication, Lecture Notes in Computer Science, 2000.