JOURNAL OF KOREA MULTIMEDIA SOCIETY, VOL. 9, NO. 12, DECEMBER 2006(pp. 1542-1551)

Replica Update Propagation Using Demand-Based Tree for Weak Consistency in the Grid Database

Ruixuan Ge[†], Yong-Il Jang[††], Soon-Young Park[†††], Hae-Young Bae[††††]

ABSTRACT

In the Grid Database, some replicas will have more requests from the clients than others. A fast consistency algorithm has been presented to satisfy the high demand nodes in a shorter period of time. But it has poor performance in multiple regions of high demand for forming the island of locally consistent replicas. Then, a leader election method is proposed, whereas it needs much additional cost for periodic leader election, information storage, and message passing. Also, false leader can be created. In this paper, we propose a tree-based algorithm for replica update propagation. Leader replicas with high demand are considered as the roots of trees which are interconnected. All the other replicas are sorted and considered as nodes of the trees. Once an update occurs at any replica, it need be transmitted to the leader replicas first. Every node that receives the update propagates it to its children in the tree. The update propagation is optimized by cost reduction for fixed propagation schedule. And it is also flexible for the dynamic model in which the demand conditions change with time.

Keywords: Grid Database, Replica Update Propagation, Demand-Based Tree

1. INTRODUCTION

Grid computing is an important new technology. Database system is needed for managing the large amounts of data on the grid. The Grid Database is the outcome of intercombination of Grid and Database technique. It becomes one of the most important resources to provide data management service in the Grid environment.

Data Replication is a significant aspect in Grid Database [1]. Every data can have many same replicas stored in other nodes in Grid Database environment. Replication can reduce the access delay and the bandwidth consumption. And replica consistency is one of the key issues in replica application. Currently, there are two groups of updating algorithms for replica consistency. One is strong consistency, and another is weak consistency.

Strong consistency must ensure that all the replicas are always in a consistent state. But it is very impractical for Grid Database system. In contrast, weak consistency is more practical, which provides less reply time and higher availability. When an update occurs on one replica, it will be propagated to other nodes by some order. This approach tolerates impermanent inconsistency among replicas. But after a certain period of time, the replicas should be in a consistent state assuredly. Accordingly, weak consistency should provide an efficient method for update propagation [2,3].

※ Corresponding Author : Ge Ruixuan, Address : (402-751) High-tech #1008, Inha Univ., Yonghyun-dong 253, Nam-gu, Incheon, TEL : +82-32-860-8712, FAX : +82-32-862-9845, E-mail : geruixuan@dblab.inha.ac.kr
Receipt date : Oct. 24, 2006, Approval date : Dec. 27, 2006
[†] Dept. of Computer Science & Information Engineering, INHA Univ
[††] Dept. of Computer Science & Information Engineering, INHA Univ
(E-mail : yijang@dblab.inha.ac.kr)
[†††] Electronics & Telecommunications Research Institute
(E-mail : sunny@etri.re.kr)
[††††] Dept. of Computer Science & Information Engineering, INHA Univ
(E-mail : hybae@inha.ac.kr)
※ This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Assessment).

There are several update propagation methods to implement replica weak consistency. One is demand based method [4]. A demand-based algorithm for fast update propagation of replicas has been presented. But the stability of this algorithm is not well even with leader election. And the additional overhead for sending message is considerable.

In this paper, a tree-based algorithm for replica update propagation is proposed. Leader replicas with high demand are considered as the roots of trees which are interconnected. All the other replicas are sorted and considered as nodes of the trees. Once an update occurs at any replica, it need be transmitted to the leader replicas first. Every node that receives the update propagates it to its children in the tree. Our proposed method optimizes the update propagation for weak consistency by cost reduction for fixed propagation schedule. In this paper, it can be seen that our algorithm performs well not only in static model but also in dynamic model by performance evaluation.

The rest of this paper is organized as follows. Section 2 gives a overview of previous work on replica update propagation. Section 3 presents our proposed demand-based trees model and algorithms. In section 4, we evaluate our method and describe the results. And the last section is the brief conclusions of this paper.

2. RELATED WORK

This chapter generalizes several existing researches about replica consistency and stresses describing the fast consistency algorithm.

2.1 Replica Consistency Service in Grid Environment

Grid system provides supports for global business, government, research, science and corporation as a basic establishment of data and computation resources management. Replica consistency service deals with keeping replicas up to date in the grid system.

A Grid Consistency Service (GCS) is proposed to synchronize replication update and maintain consistency in Data Grid [5]. And the data consistency is partitioned to several levels to adapt different requirements. In some levels, not all the replicas are always up to date. Various required locks over grid can be used to achieve different replica consistency.

Andrea Domenici generalized the emphases in replica consistency and presented how to design the replica consistency service in [6]. The component of the consistency service is described briefly. Then, he proposed a relaxed data consistency with a replica consistency service called CONStanza in Data Grid [7]. This method updates replicas asynchronously by way of changing the frequency of checking database modifications. It can satisfy database replication as well as file replication. It can keep the remote replicas lazy consistency.

2.2 The Fast Consistency Algorithm

Update propagation is an important part for replica consistency service. To satisfy more requests from clients with up-to-date data, the demand based approach should be taken into account.

Elias proposes a "Fast Consistency" algorithm which prioritizes the replicas by the demand in the distributed system [4]. This algorithm is that all the servers select the neighbor with the most demand to start a consistency session. If the neighbor has another neighbor with greater demand, the process will be repeated. This process continues until all the replicas are updated. This algorithm guarantees that most requests can be satisfied in few sessions efficiently.

Then, Elias analyzes behavior of this algorithm in a collection of replicas with multiple zones with high demand [8]. From this research, we can see that the fast consistency algorithm performs very well in a collection of replicas with only one high demand zone. However, in multiple zones of high

demand, the performance becomes poor. The low demand replicas can slow down the update propagation as barriers.

In [9], Elias proposes a leader election algorithm to avoid these barriers with low demand and generalizes this algorithm to Grid system. By this way, it can be considered that all the zones with high demand combine to form one high demand zone. But this algorithm may choose false leader. And an additional cost that the table of the IDs of leader nodes needs to be dynamically reconstructed periodically is considerable.

Furthermore, in the dynamic model, before any update propagation process performs in a replica, the chart with its neighbors' data must to be updated. That means, the replica need ask all its neighbors before every update process by sending messages. This problem also causes much additional overhead.

3. REPLICA UPDATE PROPAGATION USING DEMAND-BASED TREE

In this chapter, we introduce a demand-based tree structure for update propagation. The propagation algorithm and dynamic algorithm are presented also.

3.1 Demand-Based Tree Structure

In the grid database system, we choose some replicas with high demand value as leader nodes. Every leader node can be considered as the root of a tree. Other replicas are sorted by finding the nearest leader node. Each set of these replicas aligns by the sequence of descent demand value. Based on this sequence, these replicas can compose a tree. The leader nodes are interconnected by a bidirectional pointer by the sequence of descent demand value.

We consider a node as a candidate leader node when its demand is more than all its neighbors. Then, we define the average demand of a node d/n as a threshold, where d is the total demand of the network and n is the amount of replica nodes in the network. The candidates whose demand value is more than the threshold are chosen as leader nodes, others are abandoned. Fig. 1 shows an example of the leader choosing method.

All the other replicas are sorted by finding the nearest leader node. (See Fig. 2)

Each set of nodes including a leader node queues by the sequence of descent demand value. So, every node has a serial number as 0, 1, 2, 3…by the sequence. Obviously, the serial number of leader node is 0. Fig. 3 shows an example for the distribution of replica nodes.

The first set of nodes queue as the sequence for example in Fig. 4. Other nodes queue as the same.

Each set of nodes can compose a tree in which every node whose serial number is i. Every node has the children whose serial number is $c\times i+1, c\times i+2, \cdots, c\times i+c$, where c is the upper limit amount of children of a node in the trees. And every replica knows the information of its leader node, parent and children. The data structure is shown in Fig. 5(a).

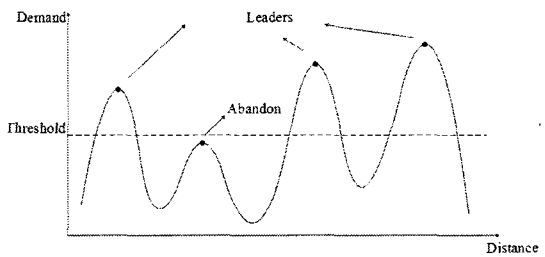

Fig. 1. Leader choosing method.

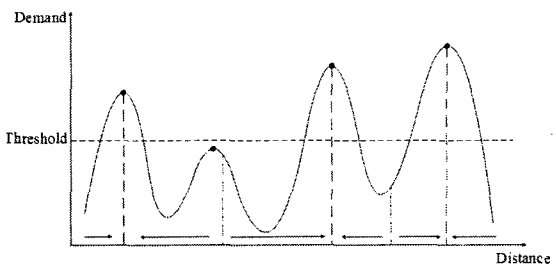

Fig. 2. Sortation of replicas.

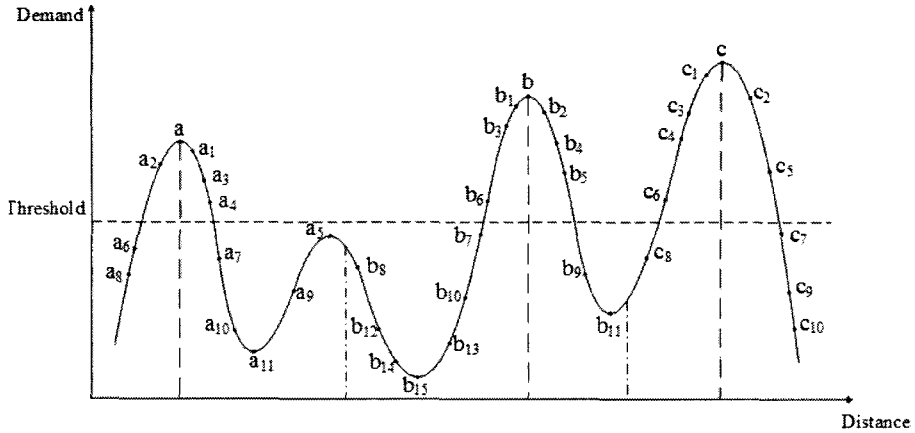

Fig. 3. An example of distribution of replica nodes.

a	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
0	1	2	3	4	5	6	7	8	9	10	11

Fig. 4. A set of nodes queue by the sequence of descent demand value

Fig. 5. (a) Data structure for tree-based structure. (b) The data for node a1.

How to decide the upper limit amount of children of a node in the trees? It lies on the amount of replica nodes in the network and the amount of leader nodes. The amount of nodes in every tree may be different. So, the height of every tree may be different. We define the minimum integer value h, which can satisfy:

$$l \cdot \sum_{j=0}^{h} c^j \geq n$$

as the average height of the trees. Here, l is the amount of leader nodes.

Based on the propagation algorithm, because we need to satisfy all the requests to clients in less time, the value $c+h$ needs to be minimal. If there are several pairs of value $<c, h>$ satisfied this condition, we choose the pair in which c is minimal for load balancing. Then, c is an approximate optimal value.

In our example, the value pair <2, 4> and <3, 3> have the minimal value for $c+h$. We choose the pair <2, 4> as the solution. Fig. 5(b) shows the data for node $\mathbf{a_1}$.

The leader nodes are interconnected by a bidirectional pointer by the sequence of descent demand value. So, the leader node also knows the information of its neighbor leader nodes in the structure. The data structure and the data for node **a** as an example are shown in Fig. 6. The tree-based structure and the example are shown in Fig. 7.

When an update occurs at a node, it will send a message to its parent initially. It can ensure that

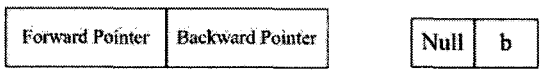

Fig. 6. (a) The data structure about neighbors of a leader, (b) The data about neighbors of node a.

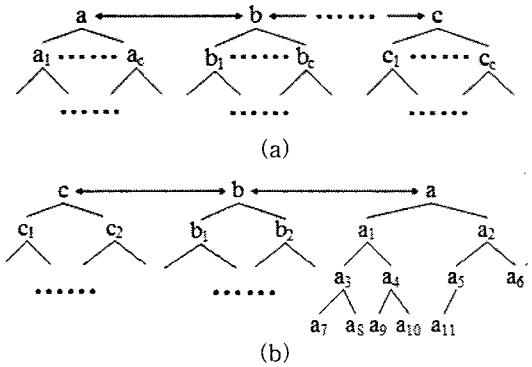

Fig. 7. The tree-based structure.

its parent won't propagate the update to itself. Then, it will transmit the update to its leader node first. The leader node, who has received the update, transmits the update to its neighbor leader node with greater demand by the forward pointer first. Next, it transmits the update to another leader node by the backward pointer. Afterward, it transmits the update to its children by the sequence of descent demand value. Every leader node transmits the update to other nodes by this sequence. Every non-leader nodes also transmits the update to its children by the sequence of descent demand value.

Fig. 8 shows the update propagation in our example when the update occurs in node $\mathbf{b}_3$.

In fact, the demand conditions do change with time. Accordingly, the tree structure need do change with time too. When the demand of a node increases, it should compare with its parent. If its demand is greater than its parent's, their position needs to be exchanged. Repeat this step, until its demand is less than its parent's. In our example, we assume some nodes with their corresponding

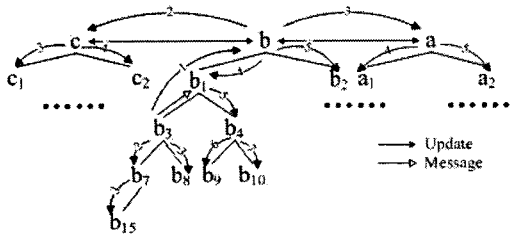

Fig. 8. The Example of the update propagation.

demand as shown in Table 1. If the demand of node $\mathbf{a}_8$ increases to 19, it should compare with node $\mathbf{a}_6$ first. So, their position needs to be changed. Secondly, it will compare with node $\mathbf{a}_1$. Their position should be changed also. Thirdly, it will compare with node **a**. Its demand value is less than the demand of node **a**. Therefore, the result is shown as Fig. 9(a).

When the demand of a node decreases, it should compare with its children. If its demand is less than any of its children's, it will exchange the position with its child with most demand. Repeat this step, until its demand is greater than its children's. In our example, if the demand of node $\mathbf{a}_2$ falls from 17 to 9, it will compare with his child $\mathbf{a}_5$ that has most demand among its children. So, it need change the position with $\mathbf{a}_5$. Next, it will compare with node $\mathbf{a}_{11}$. Its demand is greater than the demand of $\mathbf{a}_{11}$. Consequently, the result is shown as Fig. 9(b).

Table 1. The demand of replica nodes

Replica	a	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	b
Demand	20	18	17	15	14	12	11	10	8	7	5	4	25

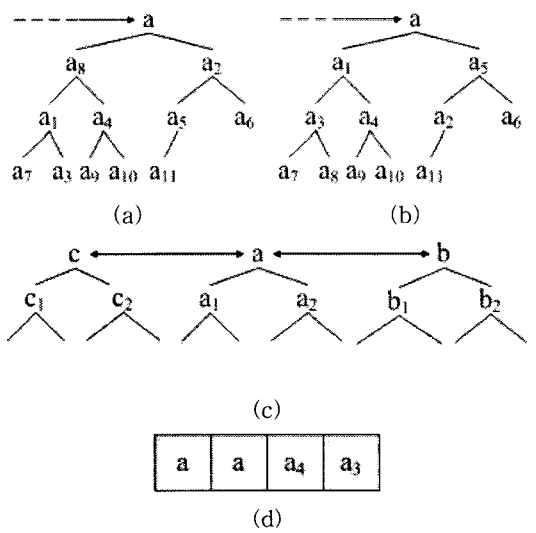

Fig. 9. The result of the dynamic algorithm used in our example.

If any leader node finds that its demand value is more than anterior one or less than the posterior one, the position of these two trees need to be changed. In our example, we assume the demand of node **b** falls from 25 to 19, and it's a leader node still, the tree including node **b** needs change the position with the tree including node **a**. The result is shown in Fig. 9(c).

When the demand of a node does change, its information for its parent should be modified. In our example, if the demand of $\mathbf{a}_4$ increases to 16, the data for node $\mathbf{a}_1$ is shown as Fig. 9(d).

3.2 Propagation Algorithm

Fig. 10 shows the update propagation algorithm after updating a replica.

If the current node is a leader, and its forward pointer is not null, the update should be propagated to the previous neighbor leader node (Line01~04) (See Fig. 11(a)). And if its backward pointer is not null, the update should be propagated to the following neighbor leader node (Line05~07) (See Fig. 11(b)). Then, it transmits the update to its children one by one (Line08~09) (See Fig. 11(c)). If the current node is not a leader node, it should send a message to its parent initially (Line10~11) (See Fig. 11(d)). Then, it propagates the update to its leader first (See Fig. 11(e)). Finally, it transmits the update to its children one by one (Line12~15) (See Fig. 11(f)).

```
Input:
current_node: Current node ID
update_data: The update need to be propagated
Variables:
FP: Forward Pointer of current node
BP: Backward Pointer of current node
child(i): the children of current node
Begin
01:  if current_node is a leader
02:    if FP is not null
03:      propagate update_data to FP
04:    endif
05:    if BP is not null
06:      propagate update_data to BP
07:    endif
08:    for i ← 1 to c && child(i) is not null
09:      propagate update_data to child(i)
10:  else
11:    send a message to parent
12:    propagate update to parent
13:    for i ← 1 to c && child(i) is not null
14:      propagate update_data to child(i)
15:  endif
end
```

Fig. 10. The update propagation algorithm.

3.3 Dynamic Algorithm

Fig. 12 describes the dynamic algorithm when the demand of a leader replica changes.

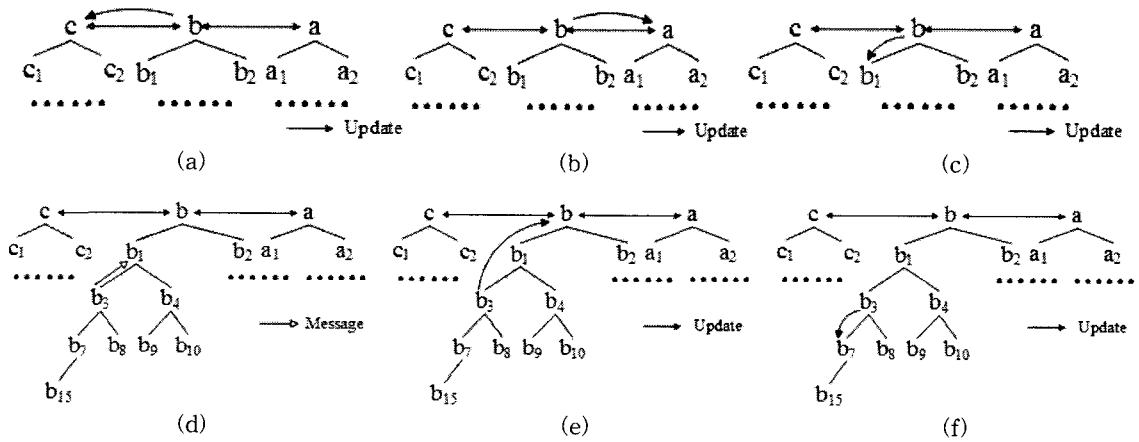

Fig. 11. Illustration of the propagation algorithm.

```
Input:
current_node: Current node ID
Variables:
FP: Forward Pointer of current node
BP: Backward Pointer of current node
current_demand: The current demand of current node
previous_demand: The previous demand of current node
first_child: The first child of current node that may
exchange the position with current node
exchange_child: The child of current node that may
exchange the position with current node
c_parent: The parent of current node
Begin
01:  if current_node is a leader
02:    if current_demand > previous_demand
03:      while FP is not null and crrent_demand is
         more than the demand of FP
04:        FP ← ExchangeLeader (FP, current_node)
05:    else
06:      exchange_child ← first_child
07     while current_demand is less than the demand
       of exchange_child
08:      exchange_child ← Exchange (current_node
         , exchange_child)
09:      if exchange_child ≠ first_child
10:        while BP is not null and the demand of
           first_child is less than the demand of BP
11:          BP ← ExchangeLeader (first_child, BP)
12:      else
13:        while BP is not null and current_demand
           is less than the demand of BP
14:          BP ← ExchangeLeader (current_node,
             BP)
15:      endif
16:    endif
17:  endif
end
```

Fig. 12. The dynamic algorithm for leader node.

If the demand of a leader node increases and is greater than the previous leader node, the position of the two leader nodes with the trees should be exchanged. This step should repeat until the demand of current node is less than the previous one (Line01~04) (See Fig. 13(a)). *ExchangeLeader* function is used for exchanging the position of two neighbor leaders. If the demand of a leader node decreases and is less than any of its children, it should exchange the position with its first child. This step should repeat until the demand of current

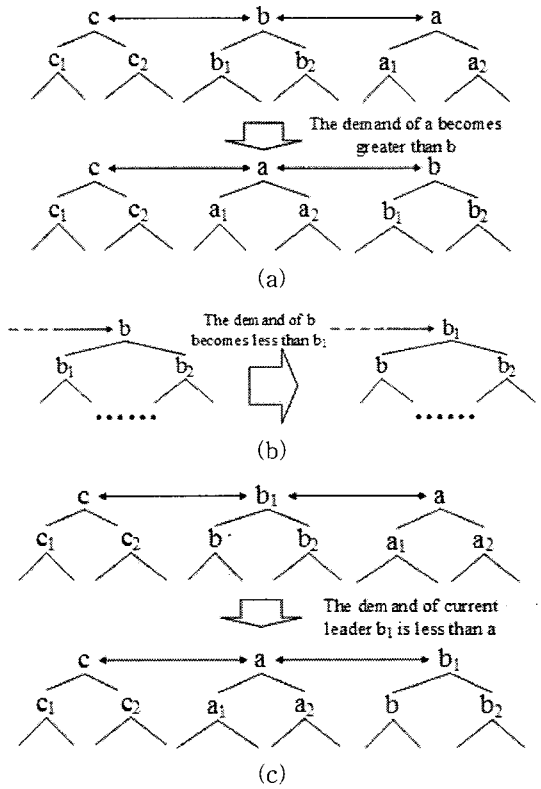

Fig. 13. Illustration of the dynamic algorithm for leader node.

node is more than all its children (Line05~08) (See Fig. 13(b)). *Exchange* function can exchange the position of two replicas with different height. Then, compare the demand of the leader after previous steps with the following leader node. If its demand is less than the following one, their position should be exchanged. This step should repeat until the demand of the leader node is more than the following one (Line09~17) (See Fig. 13(c)).

Fig. 14 is the presentation of the dynamic algorithm when the demand of a non-leader replica changes.

If the demand of a non-leader node increases, it should exchange the position with its parent (Line01~04). Contrarily, it should exchange the position with its first child (Line05~09). These steps also should be executed circularly until the node is on the proper position.

```
Input:
current_node: Current node ID
Variables:
current_demand: The current demand of current node
previous_demand: The previous demand of current node
c_parent: The parent of current node
c_child: The first child of current node
Begin
01: if current_node is not a leader
02:   if current_demand > previous_demand
03:     while current_demand is greater than the
        demand of the demand of c_parent
04:       c_parent ← Exchange (c_parent, current_
          node)
05:   else
06:     while current_demand is less than the de-
        mand of the demand of c_child
07:       c_child ← Exchange (current_nodet, c_child)
08:   endif
09: endif
end
```

Fig. 14. The dynamic algorithm for non-leader node.

4. Performance Evaluation

The evaluation environment and the comparison between the fast consistency and the demand-based tree are presented in this chapter.

4.1 Evaluation Environment

The Network Simulator (NS-2), which is an object oriented simulator, is used to provide the simulation of the network [10]. In NS-2, the whole simulation is driven by the discrete event.

BRITE [11] which is a universal topology generation tool is used to generate the topologies in our simulations. This tool generates the topologies randomly and it is inclusive, flexible, extensible and efficient.

The fast consistency algorithm and the demand-based tree algorithm are compared in the dynamic model, which is more similar to the real network. The simulations are implemented with the replicas with the number from 10 to 210. Table 1 shows the evaluation environment for testing the proposed method.

4.2 Evaluation Results

In the evaluation test, the demand of replicas changes randomly.

Fig. 15 shows the comparison of time for complete consistency between the fast consistency with leader and the demand-based tree method. We can see that the network can achieve the consistency state with less time by using the proposed method than fast consistency method.

The reason is that the proposed method only exchange messages when the demand of any replica changes. The message passing between two nodes will not affect the update propagation process. Whereas fast consistency need to send messages before each update propagation carries out for updating the chart with the data of its neighbors. So, fast consistency method needs exchanging much

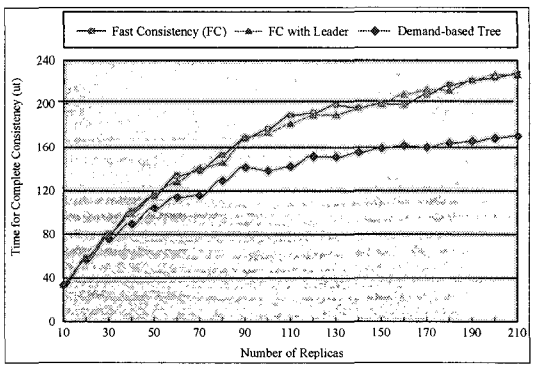

Fig. 15. Time for Complete Consistency.

Table 2. Evaluation Environment (ut: unit time)

Evaluation Element	Data Range	Evaluation Element	Data Range
Simulation Time	5000 (ut)	Transmission Time	2~4 (ut)
Number of Nodes	10~210	Update Processing Time	6~8 (ut)
Number of Clients	10~400		

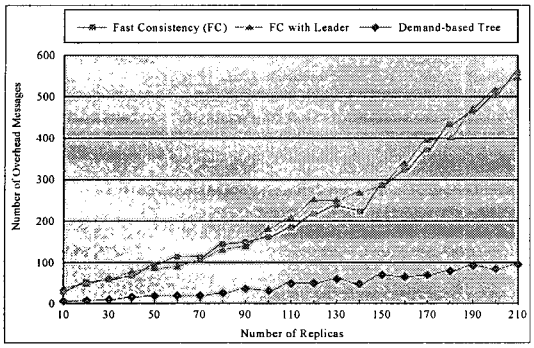

Fig. 16. Number of Overhead Messages.

more additional messages for communication between replicas than demand-based tree method, even with leader election. For this reason, both the time and the overhead of message passing in fast consistency method are more than demand-based tree method.

The comparison of the number of overhead messages is shown in Fig. 16.

Except the foregoing words about the good performance of our method, it is proved that the demand-based tree approach will not choose false leader from the evaluation. But fast consistency with leader can choose false leaders. The reason is that the votes for the election method can reach to the replicas with low demand whose neighbors have less demand.

5. CONCLUSION

In Grid Database, the data can be updated by user at any time. So, replica consistency is an important issue obviously. The demand based approach for replica update propagation is one of the methods to maintain replica weak consistency.

In this paper, the demand-based tree for replica update propagation method is proposed. The tree construction is composed based on the demand of replicas. Update on one replica can be propagated to its parent, children and the leader by this construction. Compared with fast consistency, the proposed method can achieve the consistency state with less time than the fast consistency method. It will not select false leader. It requires just little message passing. Furthermore, periodic leader election is unnecessary. The dynamic algorithm can ensure that the set of leader replicas keeps up to date. Consequently, compared with fast consistency, the overhead is reduced considerably compared with fast consistency in our method. In a word, the proposed method showed increased performance than fast consistency.

This method is suitable for Grid Database. It can also be used for various applications with large number of nodes, such as distribution system, and so on.

6. REFERENCE

[1] Houda Lamehamedi, Boleslaw Szymanski, Zujun Shentu, and Ewa Deelman, "Data Replication Strategies in Grid Environments," *Proc. Of the Fifth International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'02)*, 2002.

[2] A. Adya, "Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions," *PhD thesis Massachusetts Institute of Technology*, Department of Electrical Engineering and Computer Science, Mar. 1999.

[3] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and Demers, "Flexible Update Propagation for Weakly Consistent Replication," *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16)*, Saint Malo, France, Oct. 5-8, 1997, pages 288- 301.

[4] Jesús Acosta Elias and Leandro Navarro Moldes, "A Demand Based Algorithm for Rapid Updating of Replicas," *IEEE Workshop on Resource Sharing in Massively Distributed Systems (RESH'02)*, July 2002.

[5] D. Du¨ llmann, W. Hoschek, J. Jaen-Martinez, A. Samar, H.Stockinger, and K. Stockinger, "Models for replica synchronization and con-

sistency in a data grid," *in: Tenth IEEE Symposium on High Performance and Distributed Computing (HPDC-10)*, San Francisco, CA, Aug. 7 - 9, 2001.

[6] Andrea Domenici, Flavia Donno, Gianni Pucciani, Heinz Stockinger, Kurt Stockinger, "Replica consistency in a Data Grid," http://sdm.lbl.gov/~kurts/research/acat2003_replication.pdf, July 2004.

[7] Andrea Domenici, Flavia Donno, Gianni Pucciani, and Heinz Stockinger, "Relaxed Data Consistency with CONStanza," *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID' 06)*, 2006

[8] Jesús Acosta Elias and Leandro Navarro Moldes, "Behaviour of the fast consistency algorithm in the set of replicas with multiple zones with high demand," *Symposium in Informatics and Telecommunications*, SIT2002.

[9] Jesús Acosta Elias and Leandro Navarro Moldes, "Generalization of The Fast Consistency Algorithm To a Grid With Multiple High Demand Zones," *Computational Science, ICCS*, 2003.

[10] The Network Simulator - ns-2, http://www.isi.edu/nsnam/ns/

[11] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: Universal Topology Generation from a User's Perspective," *Technical Report BUCS-TR2001 -003*, Boston University, 2001.

Ruixuan Ge

2005 Department of Computer Science and Technology, Chongqing University of Posts and Telecommunications, China (Bachelor Degree)

2005~Present, School of Computer Science & Engineering, Inha University (Master Candidate)

Research Interests : Grid Database, Storage System

Yong-Il Jang

1997 School of Computer Science & Engineering, Inha University (Bachelor Degree)

2001 School of Computer Science & Engineering, Inha University (Master Degree)

2003~Present, School of Computer Science & Engineering, Inha University (Ph.D. Candidate)

Research Interests : Web & Mobile GIS, Database Cluster, LBS, Grid Database

Soon-Young Park

1989 Department of Computer Science, Inha University (Bachelor Degree)

1991 Department of Computer Science, Inha University (Master Degree)

2006 School of Computer Science & Engineering, Inha University (Doctor Degree)

February. 1991~Present, Senior Researcher in Database Research Team of Electronics and Telecommunications Research Institute

Research Interests : Storage System, Database System, Sensor Database

Hae-Young Bae

1974 Department of Applied Physics, Inha University (Bachelor Degree)

1978 Department of Computer Science, Yonsei University (Master Degree)

1989 Department of Computer Science, Soongsil University (Doctor Degree)

1985 Visiting Professor in Univ. of Houston

1992~1994 Director of Computing & Information Center, Inha University

2004~2006 Dean of Graduate School of Information & Telecommunication Engineering, Inha University