Object management for real time distributed database system using window algorithm with caching

Aditi Patankar and Madhumita Chatterjee

Abstract- **Real-time distributed database must maintain the consistency requirement of objects and must service the arriving request before the deadline. Users of mobile computers have online access to real-time distributed database over wireless networks. Because of limited bandwidth in wireless network, the cost for message transfer is very expensive as compared to the wire communication.**

In this paper, we present a survey of different data replication strategies and do a comparison with the model of dynamic window mechanism (DWM) algorithm jointly implemented with different types of object replacement policies. The survey shows that how dynamic window mechanism minimizes the total cost incurred for servicing of requests (both read and write).

Keywords- **Dynamic window mechanism, allocation, replacement strategies, communication costs**

I. INTRODUCTION

TODAY users of mobile computers like palmtops, notebook computers, Personal Digital Assistant (PDA) have been widely increased. As the mobility is the main characteristic of these computing machines, they have to rely on online access to real-time distributed database via wireless networks. The link bandwidth being limited, the cost of servicing arriving request (read/write) is very high and needs to be minimized.

The transactions in a real time distributed database system (RTDDBS) are random read and write requests. Servicing an object request may incur I/O costs, communication costs such as control message transfer cost and data message transfer cost. As a cost of servicing a request associated with a local database is different from that associated with the remote database, the system performance is very sensitive to the distribution of the replicas among the nodes. In order to guarantee the data consistency among multiple replicas, every change to the object must be transferred to all the available replicas in the system.

An allocation method determines whether or not the allocation scheme changes over time. In a static allocation method, the allocation scheme does not change over time, whereas in a dynamic one it does. One copy and two copies are the two possible static allocation schemes of the data item x to a mobile computer. In the first scheme, only the stationary computer has a copy of x, whereas, in the second scheme, both the stationary and the mobile computer have a copy of x.

In most of the works so far for data management [6],[7], the available local database buffer to store the replicas of the objects is assumed to be infinite. But in reality local database capacity, at a processor is of finite size. Therefore, when a processor's local database buffer is full, while an allocation and replication algorithm informs this processor of the need to save the newly requested object, we face problems like: *Should the newly requested object be saved or not? When, where and how it should be saved?*

The authors in [2] discuss a network model which tells how the objects are stored in the machines. Cost model is then presented that considers all the communication cost of the operations involved in servicing read-write requests which includes Control-message transferring cost and Data-message transferring cost [4].

Using this cost model an online algorithm called as Real-Time Distributed Dynamic Window Mechanism Algorithm is designed that can dynamically adjust object allocation schemes based on arriving read-write requests. The key idea behind the DWM algorithm is to decide the read request are saving read request or non-saving read requests. In addition, one needs to take into account the issue of buffer capacity constraints at the nodes.

In this paper, we have done a comparative analysis of the different data replication strategies i.e static allocation with DWM without buffer constraints and with buffer constraints.

II. NETWORK MODEL

The Real-Time Distributed Database system that is considered in [2], consists of n number of nodes, denoted as a_1, a_2, a_n, which forms a message-passing network for internodes communication. Each individual node consists of a processor and its local memory. All the local memories are private and are accessible by their respective processors only. Requests with deadlines arrive at the node concurrently and there is a concurrency -control mechanism to serialize them.

Two types of communication costs [4] when servicing requests are: a Data-Message i.e. object transferred between

Ms. Aditi Patankar Ramrao Adik Institute of Technology, Mumbai,Email: aditi_marathe@rediffmail.com

Ms. Madhumita Chatterjee, Ramrao Adik Institute of Technology, Mumbai,Email: c_a_mita@yahoo.com

the processors via the underlying network. A control-message transfer is needed when a node want to read an object which is not in its local memory. The notation C_c is to denote Control-message transferring cost and C_d to denote a Data-message transferring cost [1, 2].

Normally the size of a control-message is very short than a data-message. So, one can write $C_d > C_c$. In the proposed system, it is assumed that for every object o, an initial allocation scheme IAo is given by a fixed processor set S (o). The processors in S (o) are called servers. For different objects initial allocation schemes may be different. It is assumed that each processor in the system knows the server set of every object in the network. Now suppose a request for object o has arrived at the processor, then the Data processor for this object o is a processor having this object in its local memory. The other processors are called as non-data processors of object o when servicing request.

Therefore, in this system, when a processor Pi wants to read an object o, if Pi is a data processor, then object o is directly retrieved from its local memory; otherwise, since Pi knows server set S(0), Pi will send a read request to the nearest server , say Pj, in S(o). This will incur Cc units of cost. As a response, Pj will retrieve object o from its local memory and this will incur C_d units of cost. Then in order to minimize the total servicing cost of future requests, Pj may indicate Pi to save object o in its local memory. Such requests are termed as a saving-read request.

III. COST MODEL

A cost model is presented by Wujan and Huag in [1] which is used in Real-Time Distributed Dynamic Window Mechanism Algorithm to compute the cost of servicing read or write request arriving at a processor Po.

A. *Read request:*

Consider servicing a read request Ro arriving at a processor Pi with deadline d and let Ao be the initial allocation scheme of this object o known by processor Pi. Then,
Cost (Ro (Pi (d)) =

$$\begin{cases} 1 \ldots\ldots\ldots\ldots & \text{if } Pi \in Ao \\ 1+C_c+C_d \ldots\ldots & \text{if } Pi \notin Ao \ \& \ Ro\ (Pi\ (d)) \text{ is not a saving read} \quad (1) \\ 2+C_c+C_d \ldots\ldots & \text{if } Pi \notin Ao \ \& \ Ro\ (Pi\ (d)) \text{ is a saving read} \end{cases}$$

In equation (1), if Pi $\in$ Ao, then Pi must be the processor Po itself and object o is retrieved directly form its local memory. But if Pi $\notin$ Ao, then Pi sends a read request to its nearest server pj. As a response, Pj retrieves object o from its local memory and sends it to Pi.

From the above cost model, it should be noted that the only cost difference between saving read and non saving read request is 1 due to the saving operation.

B. *Write request:*

Consider servicing a write request Wo from processor Po for object o with deadline d. Then the cost of servicing this request can be given by,

$$COST_{RDDWM}\ (Wo\ (Pi\ (d))) = |A_o/A'_o|\ C_c + (|A'_o|-1)\ C_d + |A'_o|$$

Where (A_o/A'_o) are the processors in A_o but not in A'_o. A write request creates new version of an object. So, in order to maintain consistency among the replicas of an object (if there are any), the invalidate control messages have to be sent to all the processors in (A_o/A'_o), since these copies of the object o are considered to be obsolete. Thus, this is in the first term. The next part $(|A'_o|-1)\ C_d$ is the cost of transferring the new copy of the allocation scheme A'_o except pi itself. The last part accounts for the I/O cost when processors in A'_o save the object into their respective local databases.

IV. DYNAMIC WINDOW MECHANISM

Concurrency control mechanism [8] is assumed in every node to serialize the arriving request in such a way that it outputs at most one request in ∂ time units, where $\partial = 1$.

In order to service these requests on or before their deadline periods, whenever a request is released from its concurrency control mechanism, RDDWM is invoked to service this request.

RDDWM consists of dynamic window mechanism. Here, multiple request windows are generated in every processor, one for each requested object. A request window for an object o is denoted as Window (o). This request window is of FIFO type and has size equal to n, where n is the total number of requests made for an object o.

There are two counters, C1 and C2 for each Window (o). C1 has initial value n and the value of C1 is decremented by one per time unit until it reaches 0 at which window will be deleted from that processor. C2 will keep track of deadline period for requests in Window (o).

Here, it is assumed that a request sequence forms two phases[1][2]: Phase 1 and Phase 2. Phase 1 consists of several read requests only and Phase 2 consists of a write request followed by several read requests.

In fig. (a), when C1=n1, a write request arrives at Window(o), Thus, the previous three read requests in window forms phase 1. A window is closed when write request arrives at it and read requests are serviced. After servicing these read requests, Window (o) will be deleted and W(P3:d4) is inserted into new Window(o). In fig. (b), when C1=0, phase 2 is formed, as there is a write request followed by read requests. In fig. (c), since C1-0, phase 1 is formed. After servicing these requests, window will be deleted from the system.

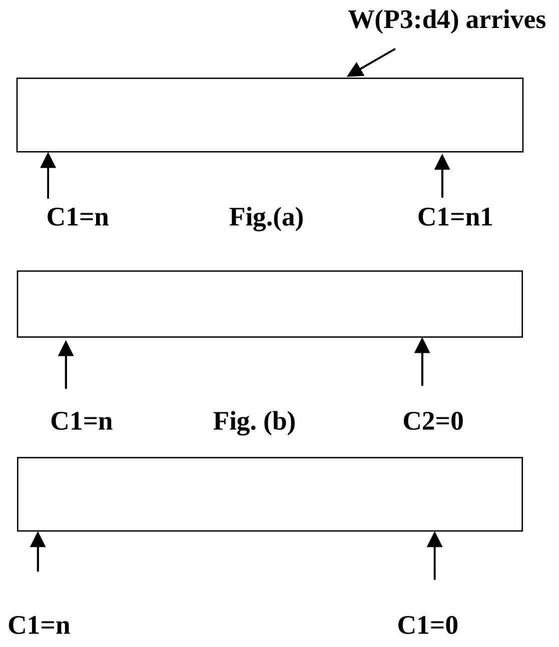

Fig.(c)

Finally, it may be noted that if the deadlines imposed by the requests is short for the system to process then there might be some requests which do not get serviced by the system i.e. that they might be dropped by the system. In this case it has been assumed that those requests will have to leave the system and have to be resubmitted.

V. SERVICING OF PHASES

A. *Type I:*

Type I phase consists of all read requests.

The mathematical model to represent total number of read and write requests is given as follows:

$$\Delta = n_r(1) -[\sum_{k \varepsilon N_a(1)/N_p(1)} n_r(1,k)-n_p(1)].(C_c+C_d)$$

The term $- \sum_{k \varepsilon N_a(1)/N_p(1)} n_r(1,k)$ represents the total number of read requests issued by processors which have the object in their respective local databases, where $N_a(1)/N_p(1)$denotes the set of processors in $N_a(1)$ but not in $N_p(1)$.

If all the first read requests issued by the processors in Np(1) are considered as saving-read requests, then the cost of servicing read requests in Type I sub schedule is less by Δ than that when these requests are considered as non saving-read requests. However, these saving-read requests will incur additional I/O operations in np(1). Besides, a write request if it exists, will incur at most np(1).Cc of additional control message cost, which comes from the "invalidation" of the redundant replicas existed in the system.

Therefore, if the DWM finds that Δ > [np(1) + np(1).Cc], each of the first requests issued by the processors in Np(1) should be considered as a saving-read request. Otherwise, if Δ <= [np(1) + np(1).Cc], then all the requests issued by the processors in Np(1) should be considered as non saving-read requests.

B. *Type II:*

Type II phase consists of write request followed by read requests.

Here,

$$\Delta = n_r(1) -[\sum_{k \varepsilon N_a(1)/N_p(1)} n_r(1,k)-n_p(1)].(C_c+C_d)$$

After the first write request is serviced, if each of the first read requests issued by processors in Np(i) is considered as a saving-read request, the service cost is less by Δ than that when those requests are considered as non saving- read requests. However, these saving-read requests will incur at most [np(i)+ np(i)].Cc amount of cost more than that when these requests are not considered as saving-read requests.

Therefore, if the DWM finds that Δ> [np(i) + np(i).Cc], then each of the first read requests issued by processors in

Np(i) should be considered as a saving-read request. Otherwise, if Δ <= [np(i) + np(i).Cc], all the read requests will be considered as non saving-read requests.

In the above discussed scheme [2], the authors assume that the available resources at the processing site of distributed database system are always plentiful like the available local database buffer to store the replicas of objects is assumed to be infinite. But, practically local database capacity is of finite size. So the mechanism would not be efficient in a real time practical situations. Bhardwaj and Lin in [1] propose some object replacement strategies to overcome the above drawback, which we discuss in the following section.

VI. OBJECT REPLACEMENT STRATEGIES

A read request from a processor pi for an object o may be served as a saving read request by the DWM algorithm. If the local database of pi has no enough space, then a decision must be made through a kind of replacement strategy, which decides whether or not the new object should be saved, or which object in use should be evicted from the local database to make space for the new object.

Object Models

Model A(Homogenous Object Sizes): In this scenerio, all objects in DDBS are of same size i.e, $|O_i|=|O_j|$, where $|O_i|$ is denoted as the size of an object i.

Model B(Heterogenous Object Sizes): In this scenerio, the sizes of the objects are different i.e, $|O_i| \mathrel{!}= |O_j|$. The object replacement strategies are more complicated in this case, since in order to make enough space for a new object, it is possible that more than one object in local database may need to be evicted by the replacement strategies.

Object Replacement Strategies

In their work Bhardwaj and Lin [1] propose three different object replacement strategies for both the model A and model

B. For model A, the following three object replacment strategies are proposed:

Strategy 1: No Replacement (NR): There is no replacement taking place when the free space of a processor is not sufficient.

Strategy 2: Least Recently Used (LRU): This algorithm capitalizes on the principle of temporal locality and evicts the object used least in the recent past on the assumption that it will not be referenced in the near future.

Strategy 3: Least Frequently Used (LFU): This algorithm uses the history of references to predict the probability of the future access.

Similarly for model B, three object replacment strategies are as follows:

Strategy 1: No Replacement (NR^{HET}) : As there is no replacement taking place when the free space of the processor is not sufficient, we note that NR^{HET}=NR

Strategy 2: Hetrogeneous Object Sizes LRU(LRU^{HET}): This policy also exploits temporal locality of reference. However, when evicting selected objects, LRU^{HET} may choose more than one object to evict because of the possibility that the size of first chosen object from the resident may be smaller than the new object to be saved.

Strategy 3: Hetrogenous Object Sizes LFU(LFU^{HET}): LFU^{HET} is the extension of LFU, using the object popularity history to predict the probability of a subsequent request.

Based on the survey, we have done a comparative analysis and arrived at the following results tabulated below.

COST COMPARISON AGAINST DIFFERENT ALLOCATION SCHEMES

Competitiveness	Static method	DWM without buffer constraints	DWM with buffer constraints
Stationary environment	1+Cc+Cd	2+2.Cc	2+2.Cc
Cd-1>0	not competitive	superior	Superior
Mobile environment	Not competitive	1+Cc+Cd	2+3.Cc/Cd
Cd+Cc<0.5	superior	Not superior	Not superior
With different no of request	-	High	Low
With different node capacity	-	High	Low

where Cc- control message cost
Cd – data message cost

The comparison shown above reveals that the performance of DWM algorithm with and without buffer constraints is same when stationary environment is considered. However in mobile environment, the DWM with buffer constraints give maximum efficiency. The performance of DWM is not as efficient as static allocation if the sum of data message and control message cost is lower than 0.5, where the lower bound competitive factor of DWM is 1.5.

VII. CONCLUSION AND FUTURE WORK

For servicing on-line requests arriving at a RTDDBS in the mobile computing environment, Dynamic Window Mechanism is capable of servicing requests with deadlines. Taking local database storage limitations into consideration, three strategies are proposed to cope with the situation when local database buffer has no available space for a new object. Then a comparison is done between static allocation methods against DWM without buffer and with buffer constraints.

It may be noted that neither mobility of hosts nor failure of nodes is considered. To consider mobility of hosts, some efficient mechanism has to be designed to transfer the request windows between servers and cost model should also be modified to include this additional cost of transferring request windows. The DWM algorithm can be enhanced to achieve dynamic fragmentation in distributed databases.

New replacement strategies can be proposed based on the importance of an object at a particular node.

VIII. REFERENCES

[1] Wujuan Lin and Bharadwaj Veeravalli "Practically Realizable Efficient Data Allocation and Replication Strategies for. Distributed Databases with Buffer Constraints", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 17, No. 9, September 2006

[2] Wujuan Lin and Bharadwaj Veeravalli "A Window-based Object Allocation and Replication Algorithm for Real-Time Distributed Database Systems in Mobile Computing Environment", *Member IEEE, IEEE CS,* Department of Electrical and Computer Engineering, 2004.

[3] Wujuan Lin and Bharadwaj Veeravalli "Object Management in Distributed Database Systems for Stationary and Mobile Computing Environments:", Network Theory and Applications(NETA) Series, vol. 12, Kluwer Academic Publishers, USA, 2003.

[4] Andrew S. Tanenbaum and Maarten Van Steen, "Distributed Systems: Principles and Paradigms", *Prentice Hal*l, 2002.

[5] George F.Coulouris, Jean Dollimore and Tim Kindberg, "Distributed Systems: Concepts and Design", *Addison-Wesle*y, 2001.

[6] A. Prasad Sistla, O.Wolfson and Y.Huang, "Minimization of Communication Cost Through Caching in Mobile Environments", *IEEE Transactions on Parallel and Distributed System*s, Vol. 9, No. 4, pp.378-390, April 1998.

[7] O. Wolfson and Y. Huang, "Competitive Analysis of Caching in Distributed Databases," IEEE Trans. Parallel and Distributed Systems, vol 9, no 4, pp. 391-409, Apr. 1998

[8] O. Wolfson, S. Jajodia, and Y. Huang, "An Adaptive Data Replication Algorithm", ACM Trans. Database Systems, vol 22, no 2, pp. 255-314, 1997