www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 4 Issue 5 May 2015, Page No. 11916-11921

Potential Of Replication In Distibuted R.T.D.B.

Er. Anup lal Yadav, Er. Sahil Verma, Er. Kavita
M-Tech
Asst. Prof. in C.S.E. Deptt.
EMGOI , Badhauli.
sahilkv4010@yahoo.co.in
Asst. Prof. in C.S.E. Deptt.
EMGOI , Badhauli.

1. ABSTRACT

Transactions in real-time databases should be scheduled considering both data consistency and timing constraints. In addition, a real-time database must adapt to changes in the operating environment and guarantee the completion of critical tasks. The effects of scheduling decisions and concurrency control mechanisms for real-time database

systems have typically been demonstrated in a simulated environment. Many time-critical applications data may be distributed among multiple sites. For example, such applications include command and control, industrial automation, aerospace and defense systems, telecommunications, banking, etc. In such applications, it imperative that the data be available to the requesting transactions at the time it is needed. In a typical distributed database, the transaction is required to access the remote data directly, at the risk of missing its deadline. Another problem can occur in such a scenario when the requesting transaction accesses the data, but it is not temporally valid. That is, its value is "out-of-date" because the transaction did not read from the most recent update. A replication algorithm creates replication transaction based on client's data requirements in a distributed real time databases. These replication transaction copy data objects to the site on which they are needed just in time for the deadline to occur. The algorithm carefully computes the parameters of the replication transactions so that we can guarantee that any requests that read data, in fact, read temporally valid data. This algorithm is designed to work in a static environment in which all object locations, and client data requirements are known a priori.

2. Introduction

2.1. Systems Specification :The following are the lists of assumptions that are made regarding the system, the system model and transaction model that are describe in this section: -

1) The system is static. That is, all distributed sites and every object on each site are known a priori. All read/write requests from clients are kept in queues, which are known a priori.
2) For each object there is one update transaction that we call the "sensor update transaction". There can be more than one transaction that updates the object, but only one is called the

sensor update transaction.

3) Each object has a local site, where it originates. Any other sites that require this object have a copy of it.
4) All the databases in the distributed system are homogeneous. All the sites in the system contain the same DBMS.
5) The period of the sensor update is always less than the temporal validity of the objects. That is, the object will be updated before it becomes temporally inconsistent.
6) Copies of objects are not accessible to transactions (Replication transaction) on other sites. That is, only the object on its origination site is accessible to be replicated.

2.1.1. System Model The model on which the Real Time Replication algorithm is based is made up of M distributed sites, data objects, and periodic requests and updates that access the data objects.

Objects. Each object in the system is defined as follows: Object = < OID, Value, Time, OV > Where

OID is a unique identifier of the object within the system. Value is the present value of the object. Time is the time at which the object was last updated. OV is the object validity, i.e. the time after which the value of the object is no longer valid.

Requests and Updates. Application requirements are specified as periodic Requests for data and Updates of data with following parameters: Request = <OID, per, rel, dl, LsiteID >,

Update = <OID, per , rel, dl, LsiteID >

Where Requests are read-only data accesses, and Updates are write-only. OID is the unique identifier if the requested object. Per is the frequency (period) at which the data is to be accessed. Rel is the release time at which the request / update should be started, dl is the relative deadline of the request/update within each period and LSiteID specifies the site at which the update/request was made.

2.1.2. Transaction Model : The algorithm produces a model with two types of transactions, local transaction and replication transaction. Local transaction: A transaction is a local transaction if all of its operations execute on the same site as the site on which the request is made.Replication transaction: A transaction is a replication transaction if at least one of its operations executes on a remote site. The following is the specification for the model of the transaction created by the RM.

Ttype < opers(OID), period, release, deadline > Where

type specifies the type of the transaction, local or replication. opers - set of operations on OID such as read, write etc., Period is the period of transaction, Release is the release time of the transaction and Deadline is the deadline of transaction in each period (relative to the period).

2.2 Replication Manager

In this section, Replication Manager (RM) takes the above parameters and creates the replication and local transactions according to the Replication

algorithm. The RM maps the system specifications to set of transactions. All the transactions are of type local or replication. For a request, the replication transaction must be finished before the start of the local transaction so that the local transaction can read the object from the replicated transaction. Also for an Update, the replication transaction must execute after the local transaction is finished. Hence replication manager also carries mapping of Request/Update transactions.

2.2.1. Transaction Mapping

Basically algorithm finds the appropriate sites for random occurrence of request / update. RsiteID is replication site and LsiteID is a local site.

Requests Mapping: There are two cases.

Case 1: If RSiteID == LsiteID

In this case Request maps to Local transaction specified as follows. Tlocal (opers(OID), period, release, deadline) Where opers(OID) is a read(OID)on local site of OID, Period, Release and Deadline are specified by Request.

Case 2: If RSiteID =/= LsiteID

In this case, Request maps to two transactions, a replication transaction and then a local transaction.

i) Replication transaction: Following are the parameters for the replication transaction.

Trep(opers(OID), period, release, deadline, exec-time) opers(OID) are read(OID) on site whose site ID is RSiteID and write(OID) on site whose site ID is LsiteID. Period is period of replication transaction. This period is in phase and equal to the period of sensor update so that the transactions will read valid data. Release is the start of the period, exec time is the total execution time of the replication transaction (i.e. exec time of read + exec time of write + network delay + preemption time).

Deadline Computation: The deadline is the crucial part of the algorithm. The algorithm carefully computes it in order to ensure that all requests always read valid data. Let d be the deadline of the replication transaction. Let N be the least common multiple of the periods of all Requests on OID and the period of sensor update and ‘n’ be the number of replication periods that should be considered for the analysis, where n is equal to N/period of replication transaction. We call ‘N’ the super period of replication transaction because after that the cycle repeats. Deadline computation is done for one full super period. The invalid interval is the interval of time during any period of replication transaction for which the object does not have the valid value associated with it, that is, the object is temporarily inconsistent (See Figure 2).

Initially the deadline is equal to period of replication transaction. Then, for each of the n periods, there are 3 cases to consider in calculating the deadline.

Case 1: If no requests are executing in the invalid interval, the deadline is unchanged. We need not care if there are no transactions executing in this interval, as no requests will be reading invalid data.

Case 2: If no request has started executing before the invalid interval but a new transaction enters at xi, where xi is any point of time in the invalid interval of ith period, then the deadline is changed to minimum (d, xi).

Case 3: If any request has started before or at OV and continues to execute in the invalid interval, then the deadline is changed to OV. Changing the deadline assures that the requests read the valid data. Note that once the deadline is changed to OV, the computation of deadline is stopped as we have reached the minimum deadline.

Once we have considered these three cases for each of the n replication transactions periods in the super period, the deadline is computed.

ii) Local Transaction: After the above replication transaction is created for a request, a local transaction is created for each request with the following parameters. Tlocal (opers(OID), period, release, deadline, exec time) Where opers(OID) is read(OID) on site of siteID. period, release, deadline are specified by Request. Exec time is execution time of opers(OID).

Updates Mapping: This section discusses in detail how the algorithm works for Updates. Again here we consider the same two cases. Case 1: If RSiteID == LsiteID

In this case Update maps to the Local transaction.Tlocal (opers(OID), period, release, deadline, exec time) Where opers(OID) is write(OID) on local site of OID. Period, release, deadline are specified by Update. Exec time of transaction is the execution time of write.

Case 2: If RSiteID =/= LsiteID

In this case, Update maps to a Local transaction and then a ReplicationTransaction. Each update local transaction causes the RM to create a replication transaction with one exception Even though each local update transaction may require a replication transaction to copy back, some unnecessary replication transactions can be eliminated.

The possible cases for eliminating the replication transactions are: a) If more than one local transaction has the same release and deadline, then only one of these local transactions needs to be copied back. b) If more than one transaction has the same period and starts at the same time, only the transaction with the least deadline (higher priority) creates the replication transaction.

2.3. Schedulable Model

In this section, it analyze and execute the transactions created by the algorithm translates requests/updates into the set of local and replication transactions, and determines whether each request/update is made on a local object or on a replicated object. We state and prove three theorems that indicate the correctness and goodness of the algorithm.

2.3.1. Theorem 1: All requests will always access temporally consistent data.

Proof: Consider a replication transaction TO that copies object O. Let d be the deadline of TO as computed by the JITRTR algorithm. Let OVi be the point in time in the ith period after which the copy of the object O becomes invalid and let P be the period of TO. O is temporally inconsistent in the ith period in the interval between OVi and d (see Figure 2). Thus the agolritm proves that no request executes in the invalid interval, then all requests access temporally valid data considering the above three deadline computing cases

2.3.2 Theorem 2: The period of the replication transaction TO must be equal to the

period of the sensor update transaction for object O in order for all requests to read valid data using our algorithm.

Proof: The second theorem tries to prove that the period of the replication transaction (PTo)) and the period of sensor update(PSuo) must equal, considering contradictory situation. Let us assume they are not equal i.e. we consider the two cases. The first case is that PTo > PSuo and the second case is that PTo < PSuo. Thus the agloritm proves that it is not possible to construct a replication transaction with the above two cases.

2.3.2. Theorem 3: The deadline assignment for a replication transaction from a request, made by the Real Time Replication algorithm, is necessary and sufficient for ensuring the temporal consistency of data.

Proof: We first prove the sufficient condition, and then we prove the necessary condition.Sufficient condition: Theorem 1 proves that requests always read temporally consistent data, which means that the deadline assignment is sufficient for ensuring the temporal consistency of data. Necessary Condition: To prove that the deadline assignment is necessary for replication transaction, algorithm takes the contradictory situation, considering the three cases mentioned above while deadline computations. This itself implies that the deadline assignment by our algorithm is a necessary condition to ensure the temporal consistency of data read by the transactions.

3. Observations

That is, given a random system specification, how often does the RTR algorithm produce a system that is schedulable, where all deadlines can be met. We also measured percentage of task schedulability to indicate how many tasks in a given system are found to be schedulable.

To implement the tests, we created a simulation algorithm as one of the module in which system specifications were randomly generated. The system specifications provided input to the RTR algorithm, and the resulting transactions were tested for schedulability using simulating module. For comparison we also simulated an algorithm for creating full replication transactions and no replication transactions.

4. Conclusions

In this paper we have presented an algorithm for replication of data transaction and simulating in a distributed real-time database.The algorithm works in a static environment in which the data available, and that guarantee that only valid data will be read.We have proven that the algorithm uses necessary and sufficient conditions for providing valid data to all requests. They indicate that the benefit of guaranteed temporal validity that outweighs the other replication strategy.

5. References

1. A. Bestavros, K. Lin, and **S.** H. Son, "Real-Time Database Systems: Issues and Applications,''Kluwer Academic Publishers, 1997.

2.A. Bestavros, "Advances in Real-Time Database Systems Research," ACM SIGMOD Record, vol. 25, no. 1, March 1996.

3. Y. Kim and **S.** H. Son, "Supporting Predictability in Real-Time Database Systems," IEEE Real-Time Technology and Applications Symposium (RTAS'96), Boston, MA, June 1996, pp 38-48.
4. K. Lam, **S.** H. Son, and **S.** Hung, "A Priority Ceiling Protocol with Dynamic Adjustment of Serialization Order," IEEE Conference on Data Engineering, Birmingham, UK, April 1997.
5. M. Lehr, Y. Kim, and **S.** H. Son, "Managing Contention and Timing Constraints in a Real-Time Database System," 16th IEEE Real- Time Systems Symposium, Pisa, Italy, Dec.
6. G. Ozsoyoglu and R. Snodgrass, "Temporal and Real-Time Databases: A Survey," IEEE Trans. on Knowledge and Data Eng., August 1995, pp 513-532.