Design of Efficient Dynamic Replica Control Algorithm for Periodic/Aperiodic Transactions in Distributed Real-Time Databases

Torky Sultan[1], Hazem El-Bakry [2] and Hala Abdel Hameed[3]

[1] Information Systems Department, Helwan University, Cairo, Egypt.

[2] Information Systems Department, Mansoura University, Mansoura, Egypt

[3] Faculty of Information Technology, Misr University for Science and Technology, 6th of October City, Egypt.

Abstract

There are many issues that affect modeling and designing of real-time databases. One of those issues is maintaining consistency between the actual state of the real-time object of the external environment and its images as reflected by all its replicas distributed over multiple nodes. An efficient replica control algorithm can contribute significantly to maintain this consistency. In this paper, a replica control algorithm for medium and large scale distributed database systems is presented. Such algorithm relies on a hybrid combination of optimistic and pessimistic replication approaches. Both the theory and implementation of the proposed algorithm are described. It maintains an independent degree of consistency for each data object because it is adaptive dynamic replication control algorithm. Furthermore, it employs some system factors to increase the chance of having an updated data item locally; avoiding remote access to meet timing constrains and achieves both availability and consistency for the replicated data as much as possible. A detailed simulation shows that the proposed algorithms can greatly improve the system performance compared to the systems either without replication or with full replication.

Keywords: *Distributed database, Real-time databases, Real-time transaction, and Replica control algorithm.*

1. Introduction

Many real-time systems are inherently distributed in nature, and need to share data that are distributed among different sites. For example, military tracking, medical monitoring, naval combat control systems and factory automation etc. All of those critical systems need data to be obtained and updated in a timely fashion [1]. But, sometimes data that is required at a particular location is not available when it is needed and getting it from remote site may take too long before which the data may become invalid. This potentially leads to large number of tardy transactions (transaction that miss their deadline).

One of the solutions, for the above-mentioned problem, is *replication* of data in real-time databases. By replicating temporal data items, instead of asking for remote data access requests, transactions that need to read remote data can now access the locally available copies. This helps transactions meet their time and data freshness requirements. In order to suite the different needs of the distributed real-time systems such as different data workloads and database specifications, multiple ways to handle the replication control and different replication schemes are proposed.

In the distributed systems, replication is seen as a cost effective way to increase availability. However, replication is used for both performance and fault-tolerant purposes thereby introducing a constant trade-off between consistency and efficiency. There are two main approaches for replication; *synchronize* (also called Active or state machine) in which a collection of identical servers maintain the same copies of the system state, client write operations are applied synchronously to all of the replicas [2,3,4]. Although this approach increases consistency of the replicated data, it increases system overhead. *Asynchronous* (also called lazy or passive) replication on the other hand where changes introduced by a transaction are propagated to other sites only after the transaction has been committed. However, this approach reduces system overhead at the expense of temporal consistency.

Replication algorithms can also be characterized according to *what* and *where* the objects are replicated. The most extreme is *full* replication in which all data items are replicated to all sites in the distributed system. The benefit of full replication is that all data are available to read locally, thus, leads to increasing performance. But this slow down the system since updating one copy creates transactions for updating all other sites, also issues like concurrency control and recovery become more

complicated. Regarding the locations and number of replicas, we can distinguish between *static* replication algorithms in which both locations and number of replicas are fixed and *dynamic* replication when the locations and number of replicas are dynamically changing according to system conditions and data needs.

In this work, a replica control algorithm for medium and large scale distributed database system is presented. This is done by trying to maintain an independent consistency degree for each data object by presenting an adaptive dynamic replication control algorithm. This algorithm is based on the entire system workload of the distributed site and increase the chance to have an updated data item locally to avoid remote access to meet timing constrains and achieves both availability and consistency for the replicated data as much as possible.

2. Related Work

In replicated database systems, copies of the data items can be stored at multiple sites, which achieve two complementary features: performance improvement and high availability. On the other hand, data replication introduces its own problems; Access to a data *item* is no longer controlled exclusively by a single site, instead the access control is distributed across each site storing a copy of the data item. It is also necessary, to ensure that mutual consistency of the replicated data is provided, in other words, replicated copies must behave like a single copy. This is possible by preventing conflicting accesses on the different copies of the same data item, and by making sure that all data sites eventually receive all updates. Therefore, major issue is the development of replication protocol/policy. The problem of finding an optimal replication scheme in a general network (i.e., a replication scheme that has a minimum cost for a given read-write pattern), has been shown to be NP-complete for the static case.

Several classifications are possible for replication [4, 5, 6, 7, and 8]. In [4] Wiesmann & Schiper distinguish five different techniques (active, weak-voting, certification-based, primary copy and lazy replication). All these techniques dealt only with fully replicated databases and need a reliable total order broadcast in order to propagate the transaction updates.

All the replication models that have been developed so far can be classified into *optimistic* replication or *pessimistic* ones [9]; in the optimistic replication, all the operations are performed locally at each node as if there were alone in the system and optimistically assumes that there is no conflict with the other replicated copies located in the other nodes. This approach will increase both response time and performance by avoiding all the remote access. But, it leads to temporal inconsistency between different nodes which require a good conflict resolution and compensation polices to eliminate this inconsistency. Other models are pessimistically avoiding conflict between concurrent operations running in the other nodes by different methods such as, global lock or primary copy in which only the primary site is permitted to update its items.

There have been a number of research papers about data replication in traditional database systems where some sporadic efforts have been made for the development of different types of protocols/policies [10, 11, 12, 13, 14], but it is not for real-time systems. In the literature there is a little replication models for real-time database systems [15,16,17], one of those models [ORDER] is found in [18] where full replication is used in medium-scale or large-scale distributed real-time database systems. It presents the ORDER algorithm that is designed to work in an environment where all data types and relations in the system are known a priori. In term of scalability, the algorithm has been enhanced to a replication algorithm called On-demand Real-time Decentralized Replication with Replica Sharing (ORDER-RS). In [19], Peddi *et al.* present a replication algorithm called Just-In-Time Real-Time Replication (JITRTR), which creates replication transactions based on client's data requirements in a distributed real-time object-oriented database. In [20] a replication protocol named PRiDe designed for optimistic replication with forward conflict resolution in distributed real-time databases was described. The model defines four phases for the replication protocol: the local update phase, the propagation phase, the integration phase, and the stabilization and compensation phase. In that model, the transactions are executed locally and the replicated items are updated as if they are alone in the system, after completing the transaction, the model check s for conflict in the other nodes and a conflict resolution policy is used to resolve it.

3. The Proposed Model for Replication in Distributed Real-Time Database

Here, a replica control algorithm used by the dynamic allocation module for replication in distributed real-time database (DoMORE) is described. DoMORE is a replication model based on increasing the probability of providing the updated data for the real-time transactions to meet their timing constrains by increasing their chance to have these data locally without the need to get it remotely from other sites. This goal can be achieved by dynamically allocating data to distributed nodes according to their access pattern. The model also allows different degree of

consistency for each data object which is dynamically calculated according to different factors (entire workload and system requirements). DoMORE model allows most of the transactions to execute and commit locally as if the transaction is executed in a local, centralized database. It uses a strict-2PL commit protocol for distributed transaction, upholding the ACID properties [21], and transaction processing is guaranteed to be serializable with respect to other local transactions. As it was mentioned, the objective of the replication model is to give the clients the illusion of service that is provided by one server and the clients have no knowledge about the data existence behind. Of course, maintaining redundant data adds overhead to the system, and this can be reduced by exploring week consistency semantics of applications. This tradeoff between consistency and system cost is the main problem of all the replication models.

Generally, consistency is a term for discussing the correctness of data in a database, the database is said to be consistence if all consistency predicates are hold. For real-time database, consistency predicates can refer to the relationships between database objects and external environment that are modeled by the database from time point of view. In a replicated system we can define three different types of predicates for consistency[22]; *external* temporal consistency which deals with the relationship between an object of the external world and its image on the database, *inter-object* temporal consistency which is the relationship between different objects or events (within a single node), and it also includes the relationship between temporal data item and non-temporal data item that depend on that item, and *mutual* consistency, which reflects the relationship between the object and its copy (replica) in different remote sites.

DoMORE employs both eager and lazy replication according to the types of database items, and it guarantees that all the transactions will read an updated valid data items and maintain both Temporal Consistency and Mutual Global Consistency [23, 24, 25]. In DoMORE, global consistency is achieved through continuously propagation and integration of updates (typically, transaction updates are propagated and integrated as soon as possible, but propagation or integration may be deferred under certain circumstances, such as if there is a transient overload).

DoMORE is also based on the concept of Virtual Full Replication (ViFuR) [26] that has been introduced in DeeDS [27, 28]. It creates a perception of full replication by ensuring that all used data objects are available at the local node so that user can interact with the database as if it is full replicated and all data are available locally. Thus this approach can take the advantages of full replication such as, reducing the resource usage compared to full replication, and transaction timeliness, simplified addressing of communication between nodes, as well as support for fault tolerance. Accordingly, the database user cannot distinguish a virtually fully replicated database from a fully replicated one.

3.1 System Model

The system model consists of a group of distributed main memory real-time databases connected by high-speed networks. It was assumed that a reliable real-time communication is maintained, i.e., any messages sent over the network is eventually delivered and have predictable transmission time [25]. The whole database is segmented on different nodes, we can define segment as a group of data objects that share properties, capturing some aspects of the application semantics, and is allocated to a specified subset of the nodes (possibly temporarily inconsistent with each other). Each segment is considered as a partition of the database and as units of allocation of replicas, which can be individually replicated based on specified replication requirements from all the clients at a certain database node. If the specification indicates that a data object will never be used by any clients on a node, it does not need to be replicated to that node, and also a certain database object may not be available at a node, but the clients at the node do not need to be aware of that, because they will never access it. This is called call virtual full replication where the client assumption of full replication of the database is still valid.

Traditional RDBMs are based on the assumption that data resides primarily on disk and in a dynamic runtime environment data might be on disk or cached in main-memory at any given moment. Because disk input/output (I/O) is far more expensive than memory access, main memory databases have been used because of the high performance of memory accesses and the decreasing cost of main memory [29,30]. Because access to main memory is so much faster than disk access, we can expect transactions to complete more quickly in a main memory system. So, in the distributed transactions that use lock-based commit protocol locks will not be held as long, thus, lock contention may not be as important as it is when the data is disk resident.

In the system, a firm real-time database model is used; tardy transactions (transactions that have missed their deadlines) are aborted. Fig. 1 illustrates the main components of the model; the monitor is periodically collects the workload data of the entire system and sends them to the admission controller. It also, checks the system performance periodically and records the statistics data about the transactions' miss ratio. The admission controller is responsible for accepting or rejecting the remote

requests from the other nodes. Transaction manager is responsible for generating local update transactions and replication transactions for their nodes. We assumed here that transactions are executed sequentially and there is no concurrent transactions, however, if concurrent transaction execution is required, the algorithm can be extended to allow concurrent transaction execution, e.g., through a locking scheme. In the next work we will consider concurrent execution of transactions and use one of the concurrency control protocol suitable for real-time systems. In the priority assignment and scheduler component, it is known that scheduling is necessary to choose an action to execute when several guards are enabled. So, it is necessary that each transaction is assigned a priority (such as prioritizing local operation), local read and update actions should have precedence over propagation and replication actions. For simplicity, in this paper, Earliest Deadline First (EDF) policy is used and transactions are processed accordingly.

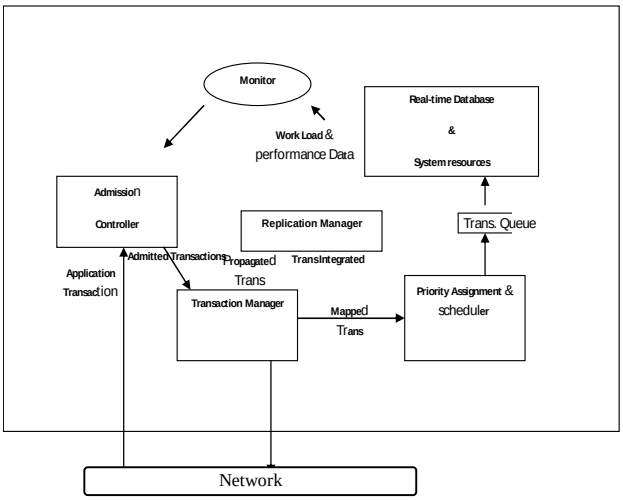

Fig. 1: The main components of the model.

As mentioned above, the model implements both eager and lazy replication, for eager replication where a synchronies replication is performed, all the sites participated in a transaction's execution are engaged in an atomic commit protocol (ACP). The model use a strict 2PL committing protocol to ensure consistent termination of distributed transactions despite site and communication failures. The two-phase commit (2PC) protocol [31] is the simplest and most used ACP. The primary goal of a two-phase commit protocol is to ensure that all participants agree on whether a transaction commits or aborts. Since 2PC consumes a substantial amount of a transaction's execution time due to the cost of its coordination messages and forced log writes to stable storage required for recovery, a number of 2PC variants appear in the literature, most notably, presumed abort (PrA) and presumed commit (PrC) [32]. As opposed to PrA, PrC has been designed to reduce the cost associated with committing transactions rather than aborting ones.

3.2 Data Model

The data model used in this paper categorizes the data used into two main categories; *Sensor* Data objects, as the real-time systems interacts with the environment through various sensors, e.g. temperature and pressure sensors and it is important to maintain consistency between the states of the environment as perceived by the sensor and with the actual state of the environment. The sensed data is processed further to derive new data called *Derived* Data that depends on past sensor data, for example the temperature and pressure information pertaining to a reaction may be used to derive the rate at which the reaction appears to be progressed which is in turn could be used to derive a new data. So we can define two different types of data items; Temporal data items that changes with time and have a validity interval and Non-temporal data items which is not change with time and thus they don't have a validity interval. If we define each site or node as a segment for a set of data items, it is called a primary site for them (Psite). For a specific data item, the copy of data item at the primary site is called *primary* copy and the copies that are replicated are called *replicated* copies or *replicas*. As illustrated in a previous work [33], the proposed database framework will be used, whereas there are two types of data located at each node; local or shared data. Local data object can only be updated by its primary site, and the shard data items can be updated by any site in the network in cooperation with its primary site. It was assumed that for each site, a backup site is predefined to be used as a backup site in case of site failure and to guarantee the minimum degree of replication.

Each replica has associated a version number (VN) which reflects the last update number for this data object. When an update is received, the receiving node (primary) increases the updating node's Version Number of local replica of the updated object. Any object has the following specifications shown in Fig.2, each $o \in O \exists o = (Id, Type, Name, PsiteId, Value, TS, VI, BUF, VN, FR)$

Id / type / name/ PsiteId / VI / BUF/ FR		
Value Current Value	TS (Time Stamp	VI Validity Interval

Fig. 2: The structure of real-time attribute.

Id: is a unique identifier for the object on his primary site.
Type: whether it is local or shard data object.
PsiteId: the object's primary site id where the object was originates, this attribute gives an indication of whether the object is a primary data object or it is a replica e.g., if PsiteId = local site, this object is a primary object

originated at this site, otherwise it is a replica for a remote data object.
Value: is used to store the final attribute value captured by the related last update method.
TS: is used to store the last time at which the attribute's value was updated.
VI: denotes object's absolute validity interval i.e., the length of the time interval following timestamp during which the object is considered to have absolute validity.
BUF: is the Basic Update Frequency, for each temporal data object it is updated periodically at a given update frequency received from its primary.
FR: A predefined freshness requirement to maintain the consistency level between different replicas scattered over all sites for the same data object.
VN: is the version number that reflects the last update number of that object.

3.3 Transaction Model

For the local type data objects, the Read Phase is performed locally at each site for only the active replicas located in this site. The Propagation phase of the replication process for any transaction updates of a local data item to remote nodes is delayed until after the transaction commits. The propagation messages for remote transactions that have been received at a node are integrated locally according to a local scheduling policy.

For the shared data item the commitment of the transaction that update it is conditional by at least the agreement of its primary site to which it is belongs using the (2PC) protocol. In that case, the integration task maintains local conflict detection data structures and is responsible for making updates by remote transactions visible to local transactions. The integration task is serialized with respect to local transactions. A transaction T and all of its updates are said to be integrated on node N if T has been committed locally and propagated to N from the other node and has been processed by the local integration task on N. The transactions are divided into read (query) transaction in which all its operations are read the data objects, while the update transaction can contain at least one write operation. Transaction can be also classified into remote or local transaction; the transaction is considered local if all its operations are performed in the local site, and it is remote if at least one remote operation. Note that only transactions of one operation are considered here.

3.4 Formal Definition

Before we describe the algorithm, we need to define some terms formally used to describe the algorithm. When a temporal data item (whether it is local or shared data item) is updated in a specific node, the *Replication Degree* ($\mathcal{RD}$) defines the number of nodes to which it must be replicated and the number of propagation messages that must be created by the Replication Manager. *Replica Allocation Set* ($\mathcal{RAS}$) defines a set of sites or nodes to which the replica updates or the propagation messages must be sent.

Definition 1: Replication Degree $\mathcal{RD}$ is the number of sites/nodes to which the propagation messages will be sent for a particular update. It is calculated by a Replica Degree Function $\mathcal{RD}$F which takes specific parameters (Node Workload, Object Freshness requirements ($\mathcal{OFR}$), User Defined Level. Note that the upper bound for $\mathcal{RD}$ is the total number of nodes in the distributed system.

Definition 2: The *Replica Allocation Set (RAS)* of a propagation transaction $\mathcal{T}_{propagation}$ = ($\mathcal{T}i_d$, *Lsite Id*, *Rsite id*, $\mathcal{WS}$, $\mathcal{GUF}$, $\mathcal{DL}$, e) is the set of remote nodes hosting replicas of objects in the write set of $\mathcal{T}$. That is:

$$RAS(T) = \{n \in N | \exists o \in nT(R(o,n) \neq \varphi$$

To determine the $\mathcal{RAS}$, the model maintains for each data object -at its primary site- a new data structure called a *needlist*, which is an array that contains a list of *sites ID* requesting that data item, and is arranged by the highest frequency rated site for demanding that object.

Definition 3: Let $\mathcal{D}$ = $(O,\mathcal{RN})$ *be* a distributed replicated database, and let $r \in R$ be the replica of object $o \in O$ on node $n \in N$. The NeedList $\mathcal{NL}$ *(o)* for o is a vector of $|\mathcal{N}|$ elements containing the latest $\mathcal{N}$ site use this object. This vector is of the form $< \mathcal{N}_1 id, \mathcal{N}_2 id, ..., \mathcal{N}_{(\mathcal{N}-1)} id >$, where each element $n, 0 \leq n < |\mathcal{N}|$ represents an identifier for a unique node or site use this data object recently.

```
void append(S_i (id)) // Append i^th Site  to the end of
the needlist.
int HighestPriority () // Returns the SiteId located at
the head of the array.
void RAF (S_i (id)) // Append i^th Site  to the RAS
void RemoveSite(S_i (id))  // Remove the i^th Site  from
the needlist after performing the RAF function on it.
// HighestPriority () and RemoveSite () both return -1
if the queue is empty.
```

Fig. 3: The methods performed in the need list.

Needlist ($\mathcal{NL}$) implements the methods in Fig. 3, the first method for adding a new site id in the need list for the intended object, this method is implemented when the object is accessed or updated by that node. When adding a new site, it is added in the head of the array. The order of the elements located in the array indicates the priorities for selecting that site to be added in the $\mathcal{RAS}$.

Definition 4: For a propagation transaction $\mathcal{T}_{propagation}$ executing in site $n \in N$ the *Replica Allocation Function RAF:* $\mathfrak{N} \rightarrow \mathcal{RAS}$ *(T)* is the function that maps a node N located in

the head of the needlist to *Replica Allocation Set* of that transaction.

As illustrated earlier, each node N hosts a set of temporal data objects as a primary site, and also maintains a set of replicas of temporal data objects hosted by other nodes. All replicas of a particular data item are updated using the fresh value from their primary copy. When a replica existed in a remote site, and periodically receives an update from its primary site within its validity interval $\mathcal{VI}$ it is called an *Active Replica*, otherwise, it is called an *Inactive Replica*. An active replica will become inactive if it is not updated within its validity interval.

Definition 5: For a set of replicas $\mathcal{R}$ of logical objects in a set O, replica $r \in R$ of a logical object $o \in O$ (where o is a sensor data object) on a particular node is called *Active Replica* $\mathcal{R}(o,\mathcal{N})$, if :

$$(CurrentTime - TS(o)) \geq VI(o).$$

Active Replicas for the derived data objects are determined according to their relative consistency, for example if we considered two objects O_1 and O_2 which have two timestamps TS_1 and TS_2 respectively, O_1 and O_2 satisfied the relative consistency called *Relative Valid Interval* $\mathcal{RVI}$ if:

$$|TS_1 - TS_2| \leq RVI$$

Definition 6: For two replicas $r1$, $r2$ where $ri \in R$ of logical objects O_1, O_2 in a set O, (where o is a derived data object) on a particular node is called *Active Replica* R(O_i,N) if: $|TS_1 - TS_2| \leq RVI$.

3.5 The Replica Control Algorithm

The goal of the proposed Replica Control Algorithm is to gain efficiency over Virtual Full Replication (ViFuR) strategy by dynamically changing the replication degree $(\mathcal{RD})$ and replica allocation set $(\mathcal{RAS})$. The main question of any replication model is *how to determine an appropriate replication level and placement for an object*? In some replication schemas the replication level for an object is predefined (e.g., 5 copies) leaving the run-time system to determine the placement of the five replicas in the network [34], while in others, it also specifies the locations of the replicas. These interfaces require the system designers to make a mapping from the desired characteristics of the (replicated) object, such as fixed level of availability, to a replication level and placement that will achieve those characteristics.

For using in this algorithm, a new replication schema is defined in which neither the replication degree nor the allocation sites is defined. Rather, for each object the replication degree and the allocation table is dynamically changing according to data access and system requirements at each site, e.g. if we have N nodes each has a set of data items (segment) that it considered as a primary site for them, the Replication Manager (RM) is responsible for dynamically calculating the replication degree $\mathcal{RD}$ that must be propagated to the other remote sites at each object update. $\mathcal{RD}$ =N-1 in case of full replication and $\mathcal{RD} \neq 0$ for fault tolerance purposes. $\mathcal{RD}$ is calculated by a separate module in the replication manager according to specific factors affecting this value (here, we consider only two factors; System workload and a predefined freshness requirement for each data object).

Using Work Load ($\mathcal{L}$) as a factor, the $\mathcal{RD}_L$ is calculated as follows:
If we have N sites to propagate a new replica, and we have 100 percentage to represent the entire workload for each node, we can divide that workload into n ranges, the difference between any two consecutive ranges is x, where x = 100/n.

$$100 - (RD_L * x) \leq L < 100 - ((RD_L - 1) * x) \quad 1 \leq RD_L \leq n \quad (1)$$

For example; if there are 5 sites in the network, and according to the entire workload, the range is calculated as follows: x = 100/5 where x=20% of workload. And when the entire workload is between 40% and 60%, then using (1) $\mathcal{RD}_L$= 3.

Because different factors can affect $\mathcal{RD}$ differently, the following weighted average equation can be used to calculate the value of $\mathcal{RD}$ to be used by the algorithm.

$$RD = ((W_1 * RD_L + W_2 * RD_{FR} + .. + (RD\ of\ m\ factors) * W_m))/m) \quad (2)$$

Where m is the number of factors and w is the weight for each factor and $(w_1 + w_2 \ldots .. + w_m) = 1$. The freshness requirement $(\mathcal{FR})$ for each object is taken as another factor affecting the Replication Degree (The $\mathcal{FR}$ is given for each object), accordingly, the $\mathcal{RD}$ can be calculated as:

$$RD = (W_1 * RD_L + W_2 * RD_{FR})/2) \quad (3)$$

The model divides the replication process into 4 phases, (Read Phase, Update Phase, Propagation Phase, and Cooperation and Integration Phase). As it was illustrated previously that data objects are classified to either local or shared data object. For the local data items, the primary site is responsible for both updating and propagating phases, while for shard data items, any site can update it, and only the primary site is taking the responsibility of the propagating phase.

Updating a replica is another decision made by the model; the primary site begins pushing replicas to the other sites when the primary site receives a new value for a specific data item from the external environment (sensor data). The

algorithm calculates $\mathcal{RD}$ as illustrated in the last section and determines the $\mathcal{RAS}$ by mapping the objects in the $needlist$ to $\mathcal{RAS}$ using $\mathcal{RAF}$ function. When the primary site pushes an update for a specific site and that data item is used by a local transaction, after committing the transaction, the site sends a need request (need req.) to the primary site which in turn add it to its need list. Note that, if the selected site use that data item one more time, it will not sends a need request unless the primary sends a new replica.

Algorithm 1 shows the pseudocode for the proposed replica control algorithm when a specific node receives a read transaction. Algorithm 2 illustrates the steps when an update transaction is received at node N. When a read transaction is received at node N, the algorithm checks for an existence of active replica(s) by checking the validity interval of the required object(s), if it exists, it will be used by the transaction, otherwise a requested transaction is created and sends to the primary site containing that object(s). Note we assume that only one site can be requested by the transaction.

If an updated transaction is received, a check for that if the requested object(s) is a primary object (Local, Shared) is maintained, as it was previously illustrated that the primary site is responsible for updating its local data objects. If the requested object(s) is a shared data object, a validity check is done, and the cooperation phase is done between this site and the object primary site. When the primary site receives the update request, it first checks the conflict existence using ($\mathcal{VN}$) of the object. And it then starts the propagation phase to other sites using the $\mathcal{RD}$ and $\mathcal{RAS}$ values generated by the Replication Manager. Transaction Manager must differentiate between the update transactions and the propagation transactions to avoid a cycle of endless propagation process, simply, a transaction type could be used.

4. Performance Evaluation of the Proposed Algorithm

A full simulation environment have been developed to test the proposed allocation algorithm, we have chosen system parameter values that are typical of today's technology capabilities, e.g., network delays. The settings for the system parameters are given in table1, while the settings for user transaction are given in Table 2. A user transaction consists of operations on temporal data objects including both sensor and derived data objects.

The transactions arrival rate follows Poisson arrival pattern, the arrival rate ג varied from (10- 80) transactions per second, accordingly, the workload applied is approximately varied from (50%-100%) when the arrival rate varied from (10-80) respectively. The execution time for one operation is between 100 microseconds to 1000 microseconds, and the transaction execution time is exponentially distributed with mean (3). The sensor execution time is uniformly distributed between (0.1 – 1) second, and the slack factor of transactions is set to 5. The Remote Data Ratio is the ratio of the number of remote data operations (operations that access data hosted by other sites) to that of all data operations. The remote data ratio is set to 20%, which means 20 percent of transaction operations are remote data operations. At each node, the entire workload varied from 20-100%.

All simulation results rely on at least ten runs, to evaluate our algorithm we use no replication and full replication as two baseline protocols. These two algorithms are the simplest, but widely used replication control strategies. The transaction miss ratios and number of messages (reflects the network overhead) of the three algorithms are shown in Fig. 4. It is clear that, among the three algorithms, the proposed algorithm gives the best transaction miss ratios under different transaction workloads.

Table 1: System Parameter Settings

Parameter	Value
Node #	10 – 50
Network Delay	1 – 3 ms
Temporal Derived Data #	200/Node
Temporal Sensor Data #	100/Node
Base Update Frequency	Uniform(0.1 - 1) sec
System Load	20-100%

Table 2: Transaction Parameter Settings

Parameter	Value
Sensor Transaction #	300
User transaction #	700
Write Operation Time	5 ms
Read Operation Time	3 ms
Slack Factor	5
Remote Transaction ratio	20%
Read/Write operation Prob.	(0.4 – 0.6) Respectively
Execution Time Of Sensor Tran.	Uniform (0.1 – 1) s
Execution Time Of user Tran.	Exp (3)
Execution Time Of Propagation Tran.	Exp (3)
Transaction Arrival Rate	(10 – 80) Trans/s

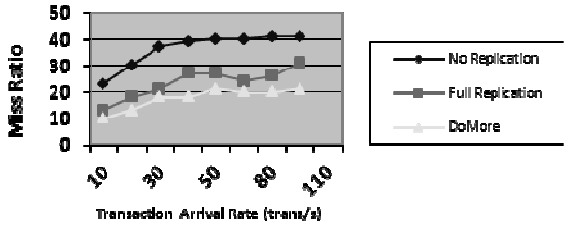

Fig. 4: Transaction miss ratio.

5. Conclusion and Future Work

A new dynamic replica control algorithm has been designed for medium and large scale distributed real-time database systems. Such algorithm has been designed to receive both periodic and aperiodic transactions while the system has no prior knowledge of its data requirements. The replicas of the data items are being dynamically allocated to the distributed nodes according to their access pattern. In addition, the proposed model allows a degree of consistency for each data object which is dynamically calculated according to different factors. A detailed simulation has shown that the presented algorithm can greatly improves the system performance compared to the system without replication or system with full replication strategy. It is desirable to enhance and extend the presented algorithm to deal with transactions of many operations instead of one operation, and deal with other parameters that affect the performance issues for distributed real-time database, such as using one of the concurrency control protocol to enable concurrent execution of transactions.

Appendix

```
//Define NeedList[n] array of nodes for each object o O : initially empty;
// LSiteId : the id of the local site where the transaction is initiated.
// RSiteId: the Id of the remote site to which the transaction is sent.
// O =(Id,Type,Name,PsiteId,Value,TS,VI,BUF,VN,FR)
// T_Read :{ (Tid,LsiteId,Rsiteid,RS,D,RGUF,DL,e) has been submitted}
Begin
    If LSiteId = RsiteId //Check if the transaction is local transaction
        Then // check if the object is primary object
        If O(PsiteId) = RsiteId
        Then Return result from executing Tread on O;
        Commit (T_Read);
        End
            Else If Current time – O(TS)<= O(VI)
            Then  //check if it is active replica;
            Return result from executing Tread on O;
            Commit (T_Read);
            End
                Else Create A remote Read transaction
                Create local update transaction;
                Return result from executing Tread on O;
                Commit (T_Read);
                End
        Else Return result from executing Tread on O;
        Commit (T_Read);
      append( (LsiteId)) // Append i^th Site  to the top of the needlist
End
```

Algorithm 1: Replica control algorithm on receiving a Read transaction

```
// Define NeedList[n]array of nodes for each object o O:initially empty;
// LSiteId:the id of the local site where the transaction is initiated.
// RSiteId:the Id of the remote site to which the transaction is sent.
// O =(Id,Type,Name,PsiteId,Value,TS,VI,BUF,VN,FR)
// T_update :{ (T_id ,LsiteId,Rsiteid,WS,RS,D,RGUF,DL,e)has been received}.
// Define RD int : replication degree  1< RD <N.
// Define  RAS[n] array of nodes for each object o O: initially empty;
// Define i int;

Begin
If LSiteId = RsiteId        //Check if the transaction is local
Then
If O(PsiteId)= RsiteId
Then // check if the object is primary object
      Return result from executing T_update on O;
      Commit (T_Read);
      VN+1; // enter the propagation phase
      For j:=1  to RD do
      Perform RAF (Si (id))  // Append ith Site located in the needlist(o)  to the RAS
      End For
      For each site S in RAS Do
      Begin
      Create  T_update  (S(id));
      End
      End For;
        Else Return result from executing Tupdate on O;
        Receive Acknowledgment from the primary site.
        Commite T_update;
Else  //if the transaction is a remote update transaction ;
      Return result from executing Tupdate on O;
      If O(PsiteId) ≠ RsiteId Then // check if the opject is not a local object;
      Commite (T_update);
      End if
      VN(o)+1;
      Else
      Send Aknowleadgment to the T_update(Lsiteid);
      append(S_i (id)) // Append i^th Site  to the end of the needlist
            VN(o)+1;
            Commite (T_update); // enter the propagation phase
            For j:=1  to RD do
                  Perform RAF (Si (id))  // Append ith Site located in the needlist(o)  to the RAS
            End for
            For each site S in RAS Do
                  Begin
                  Create  T_update  (S(id));
            End for
End
```

Algorithm 2: Replica control algorithm on receiving an update transaction

References

[1] K. Ramamritham, 'Real-time databases', International Journal of Distributed and Parallel Databases 1(2), pp. 199–226, 1993.

[2] J. Gray, P. Helland, P. O'Neil, D. Shasha, "The dangers of replication and a solution". In: Proc. of the ACM SIGMOD International Conf. On Management of Data, Vol. 25, No. 2 of ACM SIGMOD Record. ACM Press, 1996, pp. 173–182.

[3] F.B. Schneider, "Implementing Fault-Tolerant Services Using the State Machine Approach A Tutorial," ACM Computing Surveys, Vol.22, No.4, 1990, pp. 299-319.

[4] W. Matthias, S. André "Comparison of Database Replication Techniques Based on Total Order Broadcast," IEEE Trans. Knowledge Data Eng. Vol.17, No.4, 2005, pp. 551-566.

[5] F. Pedone, R. Guerraoui and A. Schiper "The Database State Machine Approach", Journal of Distributed and Parallel Databases and Technology, Vol.14, No.1,July 2003, pp. 71-98.

[6] J. Holliday, D. Agrawal, and A.E. Abbadi, "The Performance of Replicated Databases Using Atomic Broadcast Group Communication,"Technical Report TRCS99-11, Computer Science Dept.,Univ. of California, Santa Barbara, 1999.

[7] F. Pedone, "The Database State Machine and Group Communication Issues," PhD Thesis, EEcole Polytechnique Fe´de´rale de Lausanne, EPFL, Switzerland, 1999.
[8] B. Kemme and G. Alonso, "A New Approach to Developing and Implementing Eager Database Replication Protocols," ACM Trans. Database Systems, vol. 25, no. 3, pp. 333-379, 2000.
[9] Y. Saito, M. Shapiro, Optimistic replication. ACM Comput. Surv. Vol.37, No.1, pp. 42-81, 2005
[10] S. Hyuk Son."Replicated data management in distributed database systems", SIGMOD Rec. Vol.17, No.4, 62-69, 1988
[11]] O. Wolfson, S. Jajodia, Y. Huang, "An adaptive data replication algorithm", ACM on Database Systems (TODS) , Vol.22,No. 2, pp. 255-314, 1997
[12]] B. Kemme, G. Alonso, " A Suite of Database Replication Protocols based on Group Communication Primitives", In Proceedings of the The 18th International Conference on Distributed Computing Systems (ICDCS '98). IEEE Computer Society, Washington, DC, USA, pp. 156-.1998
[13] M. Wiesmann, F. Pedone, A. Schipe, B. Kemme, G. Alonso," Understanding replication in databases and distributed systems", In: Proc. 20th International Conference on Distributed Computing Systems (ICDCS 2000), Taipei, Taiwan, R.O.C., pp. 264–274, 2000.
[14] S. Cook, J. Pachl, I. Pressman, "The optimal location of replicas in a network using a READ-ONE-WRITE-ALL policy", Distrib. Comput, Vol. 15, No.1, pp. 57-66, 2002.
[15] M. Xiong, K. Ramamritham, J. Haritsa, J. Stankovic, " MIRROR A state conscious concurrency control protocol for replicated real-time databases", In Proc. 5th IEEE Real-Time Technology and Applications Symposium (RTAS 99), pp. 100–110,1999.
[16] G. Mathiason, S. Andler," Virtual full replication: Achieving scalability in distributed real-time main-memory systems. In: Proc. of theWork-in-Progress Session of the 15th Euromicro Conf. on Real-Time Systems. (2003)
[17] J. Barreto, "Information sharing in mobile networks: a survey on replication strategies " Technical Report RT/015/03Instituto Superior T´ecnico/Distributed Systems Group, Inesc-ID Lisboa, 2003.
[18] Y. Wei, A. Aslinger, S. H. Son, J.A. Stankovic, " ORDER: A Dynamic Replication Algorithm for Periodic Transactions in Distributed Real-Time Databases", In Proceedings of Real-time and Embedded Computing Systems and Applications, pp.152—16, 2004.
[19] P. Peddi, L. DiPippo, "A replication strategy for distributed real-time object oriented databases", In: Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. (2002)
[20] Sanny Syberfeldt, "Optimistic Replication with Forward Conflict Resolution in Distributed Real-Time Databases", Dissertation No. 1150, Linköping 2007
[21] T. Haerder, A Reute, " Principles of transaction-oriented database recovery", ACM Comput. Surv, Vol.15, No.4, pp. 287—317, December 1983.
[22] M. Shapiro, K. Bhargavan, Y. Chong, Y. Hamadi "A formalism for consistency and partial replication", Distributed Computing (DISC) 3274/2004, 2004.
[23] T. Gustafsson, "Maintaining Data Consistency in Embedded Databases for Vehicular Systems", Licentiate Thesis, Linköping Studies in Science and Technology Thesis No. 1138. Linköping University, 2004.
[24] S. Gustavsson, S.F. Andler , "Self-stabilization and eventual consistency in replicated real-time databases ", in Proceedings of the first workshop on Self-healing systems", (WOSS '02), Charleston, SC, USA, ACM, pp. 105—107, 2002.
[25] Gustavsson, S. F. Andler, S. (2005), Continuous consistency management in distributed real-time databases with multiple writers of replicated data", Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 2 – Vol. 03, 2005.
[26] S. Gustavsson, S. F. Andler, " Real-time conflict management in replicated databases", in 'Proceedings of the Fourth Conference for the Promotion of Research in IT at New Universities and University Colleges in Sweden (PROMOTE IT 2004), Karlstad, Sweden', Vol. 2, pp. 504–513, 2004.
[27] S. Andler, J. Hansson, J. Eriksson, J. Mellin, M. Berndtsson, B. Eftring, "DeeDS towards a distributed and active real-time database system", SIGMOD Record, Vol. 25, No.1, pp. 38–51, 1996.
[28] S. Andler, J. Hansson, J. Eriksson, J. Mellin, M. Berndtsson, B. Eftring, An overview of the DeeDS real-time database architecture" , in 'Proceedings of the Sixth International Workshop on Parallel and Distributed Real- Time Systems', 1998.
[29] J. Baulier, P., Bohannon, S., Gogate, C., Gupta, S., Haldar, "DataBlitz storage manager : Main memory database performance for critical applications", Proceedings SIGMOD '99 of the 1999 ACM SIGMOD international conference on Management of data, ACM SIGMOD Record, Vol.28, No. 2, pp. 19–520, 1999.
[30] G. Mathiason, "Segmentation in a distributed real-time main memory database", Master's thesis, University of Sk¨ovde, Sweden, 2002.
[31] R. Elmasri , S. Navathe, "Fundamentals of Database Systems", 6th Edition, Addison Wesley, 2010.
[32] C. Mohan, B.Lindsay, and R.Obermarck," Transaction management in the R* distributed database management system", ACM Trans. Database Syst. Vol.11, No.4 , pp.378-396, 1986.
[33] Torky Sultan, Hazem M. El-Bakry, Hala A. Hameed, " General Framework for Modeling Replicated Real-Time Database", International Journal of Electrical and Computer Engineering, Vol. 4, No. 8, pp. 505-511, 2009.
[34] D.L. McCue, M.C. Little, "Computing Replica Placement in Distributed Systems" A Position Paper for the Second Workshop on the Management of Replicated Data. University of Newcastle upon Tyne Appeared in the Proceedings of the IEEE Second Workshop on Replicated Data, Monterey, pp 58-61, 1992.

T. Soltan is Professor with Faculty of Computer Science and Information Systems – Helwan University, Helwan – Egypt.

Hazem M. El-Bakry (Mansoura, EGYPT 20-9-1970) received B.Sc. degree in Electronics Engineering, and M.Sc. in Electrical Communication Engineering from the Faculty of Engineering, Mansoura University – Egypt, in 1992 and 1995 respectively. Dr. El-Bakry received Ph. D degree from University of Aizu - Japan in 2007. Currently, he is assistant professor at the Faculty of Computer Science and Information Systems – Mansoura University – Egypt. His research interests include neural networks, pattern recognition, image processing, biometrics, cooperative intelligent systems and electronic circuits. In these areas, he has published many papers in major international journals and refereed international conferences. According to academic measurements, now the total number of citations for his publications is 502. The H-index of his publications is 12 and G-index is 19. Dr. El-Bakry has the United States Patent No. 20060098887, 2006. Furthermore, he is associate editor for journal of computer science and network security (IJCSNS) and journal of convergence in information technology (JCIT). In addition, is a referee for IEEE Transactions on Signal Processing, Journal of Applied Soft Computing, the International Journal of Machine Graphics & Vision, the International Journal of Computer Science and Network Security, Enformatika Journals, WSEAS Journals and many different international conferences organized by IEEE. Moreover, he has been awarded the Japanese Computer & Communication prize in April 2006 and the best paper prize in two conferences cited by ACM. He has also been awarded Mansoura university prize for scientific publication in 2010 and 2011. Dr. El-Bakry has been selected in who Asia 2006 and BIC 100 educators in Africa 2008.

H. Abdel Hameed is assistant lecturer with Faculty of Information Technology – Misr University for Science and Technology – Al-Motamayez District 6th of October City – Egypt.