Intelligent Replication for Distributed Active Real-Time Databases Systems

Rashed Salem[1], Safa'a Saleh[2], and Hattem Abdul-Kader[1]
[1]Information Systems Department, Menoufia University, Egypt
[2]Information Systems Department, Taibah University, KSA

Abstract: *Recently, the demand for real-time database is increasing. Most real-time systems are inherently distributed in nature and need to handle data in a timely fashion. Obtaining data from remote sites may take long time making the temporal data invalid. This results in a large number of tardy transactions with their catastrophic effect. Replication is one solution of this problem, as it allows transactions to access temporal data locally. This helps transactions to meet their time requirements which require predictable resource usage. To improve predictability, Distributed Active Real-time Database System (DeeDS) prototype is introduced to avoid the delay which results from disk access, network communications and distributed commit processing. DeeDS advises to use In-memory database, fully replication and local transaction committing, but full replication consumes the system resources causing a scalability problem. In this work, we introduce Intelligent Replication In DeeDS (IReIDe) as a new replication protocol that supports the replication for DeeDS and faces the scalability problem using intelligent clustering technique. The results show the ability of IReIDe to reduce the consumed system resources and maintain scalability.*

Keyword: *Replication, real-time, DRTDBS, DeeDS, clustering.*

Received February 17, 2015; accepted October 7, 2015

1. Introduction

A Real-Time Database System (RTDBS) is defined in [3] as a database system that includes all features on traditional database system, in addition to enforcing time-constraints in a form of data validation duration or transaction deadlines or both. According to [19, 20], the result value from missing deadline is used to categorize the real-time transactions into three types, soft, firm and hard. Missing a hard deadline results in an infinite penalty that has a fatal effect on the system, but missing a firm deadline gives no value, while missing soft deadline, may leave some value from the computation for some time.

Recently, the demand for real-time database is increasing. Many applications such as e-commerce, mobile communication, accounting, information services, medical monitoring, nuclear reactor control, traffic control systems and telecommunications are some examples of application that require real-time data support [2].

Most real-time systems are inherently distributed in nature, as they need to share distributed data among different sites. The data in such critical systems need to be obtained and updated in a timely fashion [16]. Sometimes, obtaining data from remote sites may take long time, which makes temporal data invalid. This leads to a large number of tardy transactions (*i.e.*, transactions that miss their deadline).

Replication is one solution of this problem. Replicating temporal data items allows the transactions to access these data locally instead of asking for these data remotely. This helps transactions to meet their time requirements [2, 9].

Predictability is the most important real-time feature of RTDBS. It is often more important than consistency, which can be relaxed to improve predictability of data accesses [3]. That is due to the fatal effect of missing deadline in a hard real-time systems and the reduced service of soft real-time systems [21]. For hard real-time systems, predictable resource usage is the most essential design concern to enable timeliness for such systems. Detailed prior knowledge about the resource requirements of transactions is necessary. Such knowledge includes the worst-case execution order of concurrent transactions, where the highest resource usage occurs [21].

To improve predictability of RTDBS, Distributed Active Real-time Database System (DeeDS) prototype is introduced [2] to provide Distributed Real-Time Database Systems (DRTDBS) with strict requirements on predictable efficient execution. However, DeeDs still needs replication features to cope with the scalability problem.

DeeDS prototype highlights how to avoid unpredictability sources from the delay that result from disk access, network communications and distributed commit processing. DeeDS also suggest some strategies to remove the unpredictability sources using in-memory database, fully replication and committing transaction locally [15]. Figure 1 summarizes DeeDS vision and strategies to improve the predictability.

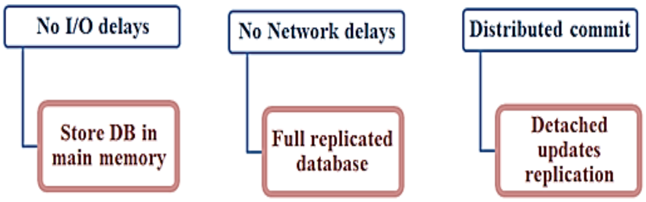

Figure 1. DeeDS vision and strategies.

However full replication has a scalability problem as it consumes the system resources because the system must replicate all updates to all the nodes, regardless whether the data will be used there or not. Also, updates in fully replicated databases must be integrated at all nodes [6, 22].

In fact, scalability is increasingly very important target for distributed real-time databases, because the number of users, the number of nodes, the involved workload, and the size of database in distributed applications are in increasing [11]. Full replication strategy of DeeDS to avoid network access delay makes systems scale badly as they replicate all updates to all nodes, using excess resources whether they are needed there or not. In real systems, not all nodes use all data objects. It is assumed that only a fraction of the replicated data is used at the local node [4].

The core design of DeeDS is extended in DeeDS Next Generation (DeeDS NG) [6] to meet the demands of modern real-time applications such as sensor networks, and information fusion-based applications. The new DeeDS design is introduced by adding the concept of virtual full replication to manage the resources for scalability [16]. It reduces the resource usage by avoiding un-needed replica by replicating selected data items to selected nodes depending on the fact that not all nodes need all data items, *i.e.*, replicate each item where it needed only.

Under the assumption by DeeDS NG that the replicas of data objects will only be used at a limited subset of the nodes (as in partial replication), the degree of replication can be lowered. The usage of bandwidth, storage and processing time depends on the degree of replication, so these resources are wasted in a fully replicated database compared to in a lower degree of replication. With replication of only those data objects that are actually used, resources can be saved and thereby scalability can be improved [2].

Virtual full replication is based on the idea of lowering the replication degree. Clustering distributed database nodes into number of clusters with lower number of nodes can be considered as an approach to limit the degree of replication and can achieve the virtually fully replicated database [5]. The degree of replication is a result of allocating node to the clusters where its data objects are accessed. This is typically much fewer nodes than used in a fully replicated database. The clustering method must be based on the accessed data and the network properties [6, 22].

The main contribution of this work is introducing Intelligent Replication In DeeDS (IReIDe) as a new replication protocol based on virtual full replication using intelligent clustering technique to support DeeDS in maintaining scalability problem. The idea of this work is that the concept of virtual full replication can be achieved by using clustering technique which segments large number of nodes based on a priori known data needs and network properties into many clusters with smaller number of nodes. This lowers the replication degree, reduces the amount of replicated data to be transferred among less number of nodes and increases the transaction processing performance. Moreover, this improves scalability by controlling the usage of the key resources [22].

Determining associated nodes based on the timing properties of the used data in addition to the timing properties of the network (communication cost) will reduces the system resource consuming, and allowing parallel replication that improves the consistency and gives more chance to increase the scalability without the headache of performance. This work tries to make the DeeDS more scalable in an intelligent way by managing resources for scalability using clustering technique to replicate updates only for those data objects that are used at a node.

Although IReIDe is introduced to keep the properties of DeeDS, it is designed mainly to avoid unpredictability, so it can be adapted to other database models.

The remainder of this paper is organized as the follows: section 2 discusses the previous work that related to the present work. Section 3 presents IReIDe approach for intelligent replication in DeeDS. Section 4 covers the experimental study to evaluate the introduced algorithm. Finally, the conclusion is provided in section 5.

2. Related Work

Although there are many researches that worked in the field of RTDB replication [3, 6, 7 , 12, 14, 17, 19] few of them addressed the replication in DeeDS or even the concept of virtual full replication [14, 15].

Using term of scalability, there are many works in different areas act to minimize the communication cost as the work by Lin and Veeravalli [12]. They used prior knowledge of accesses for minimizing service cost for allocating objects in a distributed system. They introduced a Central Control Unit (CCU) to serialize allocation and deallocation decisions. But, this control unit becomes a single point of failure and a bottleneck for scalability.

The work by Mathiason *et al*. [14] is the main work that supported the virtual full replication in DeeDS by grouping data objects into segments. The data in each segment are similar in some combination of properties. Their scalable algorithm (ViFuR-S) handles the problem of segmenting a database based on multiple properties, where multiple and overlapping segmentations of the database are allowed. Another work by Mathiason *et al*. [15] extends the previous algorithm to maintain scalability at execution time, to improve the access time to data object at any node and to ensure scalable propagation of updates over the network. Their algorithm is called ViFuR-A as it was to adapt replication over time to actual data needs of database clients. Their results showed the ability of virtual full replication by adaptive segmentation to maintain scalability and preserves transaction timeliness.

The last two works [14, 15] are the most related work to our proposed protocol as they are based on DeeDS prototype. However, we noticed the following shortcomings that will be avoided in the present work:

1. Their algorithm depends on the properties such as access location, consistency and storage medium. But timeliness which is the main property of RTDB ignored in their work.
2. Their work missed the usage of network properties.
3. They ignored the usage of pattern detection in their work.
4. They determined and fixed the replication degree of each segment, although it is difficult to occur.

We see that the replication degree is a dynamic concept that based on the requirement of each node.

By thinking of clustering the database nodes, we find that many clustering techniques have been introduced in a wide variety of applications such as image processing [17], network segmentation [23], marketing [13], pattern recognition [25], and network sites clustering [6, 10]. But in case of clustering the database nodes, the work by Hababeh [6] must come to front. He introduces an intelligent clustering technique that segmented the distributed database network sites into disjoint clusters according to the least average communication cost between network sites. This work has succeeded to reduce the communication cost. We believe that, this reduction can improve the predictability of DRTDB. However, we need to extend their clustering algorithm to include the timing features of RTDB in clustering criteria in addition to the network properties to obtain better predictability of DeeDS. This can reduce the communication traffic and improve the performance of DRTDB. Bearing in mind, the nature of time-constraint database to achieve the virtual full replication in DeeDS, we still need to develop a novel clustering method to outperform the current clustering techniques. This novel clustering method creates disjoint clusters according to the properties of data and the properties of network together to generate an optimal number of clusters. This lowers the replication degree and helps RTDBS to meet the timing requirements and to avoid the fatal effect of missing the deadline in case of hard transactions [18].

3. IReIDe Approach

In virtual full replication, every node has an image of full replicated database, and the updates will be replicated where they are needed. To achieve virtual full replication, we introduce a novel clustering method, which uses the timing properties of data and the timing properties of the network to segment the database nodes. It aims mainly at reducing the replication degree to preserve the system resources and enables DRTDBS to meet their time requirements. Such requirements of prior knowledge about the needs of each node must be used in the intelligent clustering approach. Each cluster has its own replication degree that is lower than in case of full replication. This acts to reduce the usage of system resource and make the database more scalable. A cluster is represented as a pair $\{O, N\}$ where N is a set of nodes that host a replication set which list the associated nodes for each object in data objects O.

The clustering algorithm is presented in the following subsections. The produced clusters are allowed to be overlapped but they are prevented from intercommunication between each other.

Each database node holds a specific data store called "*replication set* (*Rep_set*)" which is a Boolean two-dimension data structure to define the shared nodes (subscribers) of each data object which are together in same cluster. It is used by propagation process to distribute the updates of an object to all listed nodes for the individual object. The replica or updates are stored in another data structure to be used by the integration process.

The database system in each node uses its replication set to replicate any update on data object to the subscribers that are defined in replication set for the individual data object. The replication operation will execute in $O(n)$ where n is the number of subscribers for each data object cluster.

3.1. The Clustering Method

The decision of clustering is based mainly on the rule that the communication cost between two sites will never affect the validity of replicated data, *i.e*., any data object must be replicated within its validation period. The *allowed data* are those that have timing properties values is less than the network time communication cost. To maintain and simplify the work, the following assumptions are considered:

1. The communication cost between each two sites is symmetric.
2. The transaction type is hard, *i.e.,* it is important to avoid the missing its deadline.
3. The transaction deadline is derived from the used data *validation duration* property.
4. The database is virtually fully replicated, so any transaction that initiated on any node will access a

needed data object locally.

5. The communication is real-time reliable, *i.e.*, all messages are sent to the destination within limited time and they are reaching in save time.
6. The network is fully connected, *i.e.*, it is possible to communicate in a point-to-point fashion between any two nodes in the system.

The outline of these methods is depicted as the following:

- *Inputs*: Log matrix Objects by Transactions (OT), Communication cost matrix CC[*N*][*N]*, Total Number of Nodes (*N*), Total Number of data Objects (*O*).
- *Processing*: {Module 1: Obtaining Minimum Deadline Matrix of shared data}.
  {Module 2: Obtaining replication set/each object/each site}.
- *Output:* Matrix of generated clusters (replication set) for each site (RepSet[][][]).

3.1.1. The Definition of Clustering Parameters

The following list defines and describes the parameters of our proposed algorithm:

- *Communication Cost matrix CC (S_i, S_j)*: holds the cost of transmitting data object in *ms*/*byte* between any two nodes S_i and S_j in the distributed system.
- *Log matrix (OT)*: is the working set (the objects used by transaction) for all transactions at each *S*(*i*).
- *Deadline[O][N]*: a source of minimum deadline or validity duration of the transaction on each node.
- *Replication set (RepSet[N_c][O][N_i])*: a place to identify the node cluster of each object of the current node (N_c) on each node (N_i).

3.1.2. Module 1: Obtaining Min Deadline Matrix

This module checks the transaction DeadLine (DL) and Validity Duration (VD) for each transaction in the log matrix OT to produce Minimum Deadline matrix of each data object.

3.1.3. Module 2: Obtaining the Replication Set/ Site

This module which is presented in Algorithm 1, identifies the sites that match the time cost, which is less than the minimum deadline of shared data object between them in order to group them initially in one cluster.

```
Algorithm 1: Obtaining Replication set/ site
Inputs ➔ Deadline[O][N], Total number of Nodes (N),
Total number of Objects (O), communication cost matrix
CC[N][N]
Set Boolean matrix:Rep_set [nodes][objects][nodes]← false;
  For each row ∈deadline[O][N] using iterator (i) do:
      Set ObjectID ← i
      Set counter C ← 0
      For each col ∈deadline[o][N] using iterator j
         IF (deadline[i][j] ≠ Φ) then
             set VD= deadline[i][j]; set NodeID =j;
            set temp[C++] ←   NodeID; // temporary structure
         End if
         Loop
         For  x from 0 to C-2
           For  iterator y from x+1 to C-1
             IF CC[temp[x]][temp[y]]<VD then
           Rep_set[temp[x]][ObjectID][temp[y]]←T; // true
             Rep_set[temp[y]][ObjectID][temp[x]]←T; //true
             End if
           loop
         loop
    loop
Outputs ➔ Replication set (RepSet[N][O][N])
```

3.2. The Propagation Method

The propagation method consists of two phases: distribution phase, and receiving phase. When a transaction that accesses an object for write operation has committed, the "*distribution module*" or the *distributor* will be called. It starts with checking the *Rep_Set* to extract the *subscribers* of updated object. Then it distributes the update to the specified subscribers.If the current node is not listed in replication set as a *subscriber* for the current data object, the *distributor* will send a broadcast message with a request to update the replication set. The message contains the current object identifier, the updated value and current node identifier in addition to the timestamp value. All will be used by the receiving nodes to add the received *node_id* to its replication set for *object_id*. Also, the receiving node uses the received value to update the object value in replication directory. The *replication directory* (*Rep_DIR*) is a data structure that holds information about replica of data objects. The *distribution* module is outlined as in (Algorithm 2). Each node contains a data store which is called "*Active_list*" to hold the most recent node identifiers that access each data objects, sorted according to the timestamp of activity. The most active node is the node that has the most recent access to the object. Receiving the update message results in calling of *active_list* update module (Algorithm 3) to update the local *active_list* data store. "*Active_list*" is used to make a decision of removing the least active node when the number of subscribers increases to a specified threshold (70% in our case) of the total number of nodes. The node that receives the propagated message will call the receiver module which executes *B_receiving* module (Algorithm 4) in case of broadcast message or executes *R_receiving* module (Algorithm 5) in case of regular update message according to the message type. *B_receiving* module calls "*Add*" module after checking whether the current node is a subscriber for the received data object to add received *Node_ID* as a new subscriber. Then, *active_list* update module is called to refresh *active_list* data structure. *B_receiving* module also use the received information to update the *Rep_DIR*. In case of regular received message, *R_receiving* module (Algorithm 5) is called to update *Rep_DIR* using the received data. Then call *active_list* update module.

Algorithm 2: The process of distributor (distribution module)

```
Inputs→ current node, current object, Rep_set[O][N]
//Check Rep_set for current object's subscribes in TempList[]
Set Value← current object value
Initialize temp vector: Osubscribers[]
For each node ∈Nodes do:
        If Rep_set[current ObjectID][ node] = true then
           Add node to Osubscribers []
        End if
Loop
IF Osubscribers.size() ≠ 0 then
   /* Regular propagation to all subscribers: Multicast*/
For  each Osubscriber ∈Osubscribers[]using iterator( i ) do:
   Msg.Head← "Regular updates"
   Msg.IP← { Osubscribers [].get_IP_address }
   Msg.data←{current node_ID, current Obj_id, value, TS}
        Network.send(Msg)
Loop
Else
   /* Broadcast request to add new subscriber */
   Msg.Head← { "New subscriber" & current Obj_id }
   Msg.Address← IPAddress.getAny()
   Msg.data← {Current_node_id,current_Obj_id, value, TS}
   Network.send(Msg)
End if
```

Algorithm 3: Active_List_update module

```
Inputs→ current objectID , active_list
For each row ∈ Rep_Dir using iterator(i) do:
  Set j to 0;
  IF Rep_Dir[j][object] = objectID then
    temp[j][node]= Rep_Dir [i][node]
    temp[j][TS]= Rep_Dir [i][TS]
    increase j with 1 //j++
  End if
Loop
For each row ∈ temp using iterator(i) to temp.size-1 do:
  For each row ∈ temp using iterator(j) to temp.size-1 do:
    If temp[i]>temp[j] then swap
   Loop
Loop
   //Extract highest 7 nodes to active_list[current object]
For each row ∈ temp using iterator(i) to7 do:
Active_List[objectID][i]=temp[i]
Loop
 Outputs→ Updated active list
```

Algorithm 4: The process of B_receiver

```
Inputs→ current node, current object, Rep_set[O][N]
      Set size ←Rep_dir.size()
   // Check  Rep_set  for ObjectID in TempList[]
       For each col ∈Rep_set [ObjectID][ col]
       IF Rep_set [ObjectID][i] ≠ ϕ then add it to TempList[]
         Loop
IF TempList.size> 0 then  // So, current node is a subscriber of
object
   Call ADD method to add the received NodeID in Rep_set
   //Update Rep_dir[][] to add new record for new update
   replica
   IFRep_dir[size][ Node]  =  NodeID  then
     Rep_dir[size][ object]←    ObjectID // same with
   Active_list
     Rep_dir[size][ TS] ←  Timestamp // same with Active_list
     Rep_dir[size][ value]  ←   value // same with Active_list
   End If
   //Acknowledge sender to update its subscribers for object ID
     Msg.Head←   "add me"
     Msg.Ip← { NodeID. get_IP_address }
     Msg.data←  { ObjectID, current NodeID}
     Network.send()
   //Call Active_list updatemodule
          Call Active_list_update(ObjectID)
End if
```

Algorithm 5: The process of R_receiver process

```
Inputs→ NodeID, ObjectID, Rep_set[O][N]
//Update Rep_dir[][] to add new new update replica
Rep_dir[size][ object]← ObjectID
Rep_dir[size][ Node] ← NodeID
Rep_dir[size][ TS] ← Timestamp
Rep_dir[size][ value] ← value
Call to update Active_list[] for sent objectID
```

4. Experiments and Results

The experimental analysis to evaluate the proposed replication protocol is conducted over the DRTDBS sites. The results obtained from the experimental tests confirm that the proposed approach is abstracted from database model and independent on network topology. So it can be implemented in different DDBS environments even for the network with larger number of sites. For simplicity, we use a complete connected network consists of 10 nodes supported by a full-distributed database over different areas. The database consists of 100 data objects with time constraint properties on data level (validity duration) between (1-2.8 sec). For simplicity again, each data object contain only one time-constraint attribute.

All nodes are used as servers that execute and develop the database transactions; also they are used as terminals which produce administrational and functional information. The communication costs between sites, which are measured in *ms/byte* is presented in Table 1.

Table 1. Communications cost matrix: CC[N][N].

	Node (1)	**Node (2)**	**Node (3)**	**Node (4)**	**Node (5)**	**Node (6)**	**Node (7)**	**Node (8)**	**Node (9)**	**Node (10)**
Node 1	0	0.1	1.1	0.6	0.4	0.8	1.5	1.6	0.4	0.5
Node 2	0.1	0	1	1	0.2	0.3	0.4	0.2	0.7	0.6
Node 3	1.1	1	0	0.6	0.6	0.7	0.8	0.7	0.8	0.2
Node 4	0.6	1	0.6	0	0.8	0.7	0.6	0.7	0.1	0.1
Node 5	0.4	0.2	0.6	0.8	0	0.1	0.2	0.2	0.7	0.9
Node 6	0.8	0.3	0.7	0.7	0.1	0	0.1	0.2	0.6	0.8
Node 7	1.5	0.4	0.8	0.6	0.2	0.1	0	0.4	0.7	0.6
Node 8	1.6	0.2	0.7	0.7	0.2	0.2	0.4	0	0.8	0.7
Node 9	0.4	0.7	0.8	0.1	0.7	0.6	0.7	0.8	0	0.1
Node	0.5	0.6	0.2	0.1	0.9	0.8	0.6	0.7	0.1	0

A customized transaction log file is generated with information about each executed transaction in each node. This log contains information about the identification of used Object, and the time properties of used data of writing transactions. These log files are collected from all nodes into one matrix called OT that is needed by module 1. Processing the module #1 results in the deadline matrix. This matrix determines the nodes, which accesses each real-time data object and defines the minimum deadline of each.

The module #2 uses the previous matrix to determine the nodes which are *subscribers* of each data object and store them in a temporary list. These lists are used later by the same module to compare the communication time cost versus the deadline property of the real-time objects to make the decision of identifying the nodes that can transmit the data object within its validation period. The *Rep_set* matrix or *subscribers* reflects the result of decision from this module. Table 2 shows a sample of *Rep_set*/*subscribers* of node #1.

Table 2. Replication set for node 1.

OID	N2	N3	N4	N5	N6	N7	N8	N9	N10
43	FAL	FALS	True	FALS	True	FALS	FALS	True	FALS
30	FAL	FALS	FALS	True	True	FALS	True	FALS	FALS
48	FAL	FALS	FALS	FALS	True	FALS	FALS	FALS	True
37	FAL	FALS	True	True	True	FALS	True	FALS	FALS
92	FAL	FALS	FALS	FALS	True	FALS	FALS	FALS	FALS
65	FAL	FALS	FALS	FALS	True	FALS	FALS	FALS	FALS
4	True	True	FALS	FALS	True	FALS	FALS	True	FALS
64	True	FALS	FALS	True	FALS	FALS	FALS	FALS	FALS
47	FAL	FALS	True	FALS	FALS	FALS	FALS	FALS	FALS
86	FAL	FALS	FALS	FALS	FALS	FALS	FALS	True	FALS
34	True	FALS	True	True	FALS	FALS	FALS	FALS	FALS
7	FAL	FALS	FALS	True	FALS	FALS	FALS	FALS	FALS

To evaluate the performance of IReIDe, we measure the consumed system resources (*i.e.*, bandwidth and storage) by IReIDe compared with full replication.

In case of storage requirements evaluation, Table 3 presents the quantified storages that are used by some clusters in case of using IReIDe and without it, *i.e.*, full replication where the storage media is consumed on all nodes for all objects. Notice that a configuration of 10 nodes and 100 database objects on each node, each with a size of 64 bytes, and a replication degree of 10 in case of full replication.

Table 3. Storage requirements evaluation Of IReIDe.

# of Nodes/ Cluster	Storage cost / bytes
2	128
3	192
4	256
5	320
6	384
Full replication	**640**

While the full replication consumes a fixed large storage spaces, IReIDe success to keep the storage at low level. It can be noticed that IReIDe can reduce the total storage requirements by around 80% in case of smallest cluster or 40% in case of largest cluster. Figure 2 depicts this evaluation in two cases.

In context of the number of replica, each node has its specific number of replica depending on the replication degree of its shared objects. The maximum number of replicas that can be carried is 123 replica of 42 objects as in case of node 8. Full replication needs to store 1000 replicas per node. For comparison, we consider one case that is used by Mathieson *et al*. [15], where the accesses within a fixed number of replicas are 300. This case was not the worst-case as they reported. Comparing with 123 replica (worst case in the present approach) results in that IReIDe uses less storage cost for the number of replica.

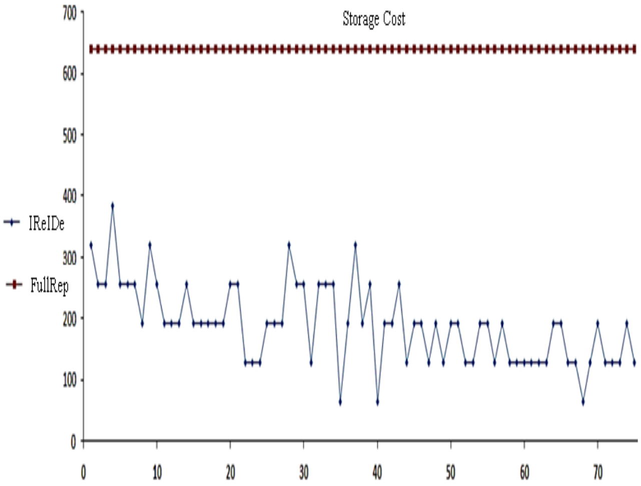

Figure 2. Storage requirements evaluation of IReIDe.

Comparing the impact of the present work to the most related work by Mathieson *et al*. [15], we find that they measure the storage need for increasing number of nodes compared to full replication, and they discover that their approach in some cases consume 12-15% more storage. In contrast to our results which achieved very high reduction to storage (around 80%), although we use the same number of nodes and the same number of data objects with same size.

Other factor to evaluate IReIDe is the consumed bandwidth. We assume that every update of data object uses one network message, so update messages are equal in size. Table 4 presents the used communication cost between each two nodes using IReIDe according to the total number of shared data objects between each two sites. The total communication cost that consumed if all replica updates occur is (290 *ms*/*byte*) in case with IReIDe while the corresponding value in case without IReIDe is (1479 *ms*/*byte*). In other words, IReIDe has succeeded to reduce used bandwidth with around 80%.

Table 4. Communication cost for replication With IReIDe.

	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10	Total
N1	0	0.5	2.2	3	2.4	6.4	1.5	3.2	1.6	0.5	**21.3**
N2	0.5	0	6	5	0.8	2.1	2.4	0.8	1.4	1.2	**24.2**
N3	2.2	6	0	4.2	4.8	4.2	5.6	3.5	5.6	0.8	**36.9**
N4	3	5	4.2	0	4.8	4.9	3.6	4.9	0.6	0.3	**31.3**
N5	2.4	0.8	4.8	4.8	0	0.9	2	2.8	2.1	2.7	**23.3**
N6	6.4	2.1	4.2	4.9	0.9	0	1	3.2	6.6	5.6	**35.7**
N7	1.5	2.4	5.6	3.6	2	1	0	6.4	6.3	3	**63.3**
N8	3.2	0.8	3.5	4.9	2.8	3.2	6.4	0	8	4.2	**34**
N9	1.6	1.4	5.6	0.6	2.1	6.6	6.3	8	0	1.6	**35.2**
N10	0.5	1.2	0.8	0.3	2.7	5.6	3	4.2	1.6	0	**19.9**

To make a comparative evaluation against

DYFRAM approach [8] which is similar to the present work, we use histograms to record some statistics about running of IReIDe approach. Although, they use four workloads to evaluate their work with, we are only considered with the first two workloads to compare with our results because their settings are closed to the settings of the present experiment. To measure the transmission reduction, the histogram records about the access rate by IReIDe and without it (full Replication case) is used. With access rate 3190 by IReIDe compared to 4000 by DYFRAM, the transmission reduction by DYFRAM reaches to ≅ 41% in general case and 52% in optimal situation. While, the transmission is reduced by IReIDe varies from 40% within the largest cluster to 80% within the smallest cluster by an average of 60%.

Freshness/Tardiness (FIT) approach [24] is similar to the present work as it tries to keep the scalability in massive distributed data by reduction of communication cost. The authors evaluate the scalability of this mechanism in compared with ODH [1] in term of the average of tardy transactions to the throughput (operations/sec). To make a comparative evaluation against FIT mechanism and its related works, we initiate a number of randomly updates (write transaction) on selected data objects in all nodes and using the performance monitor to record the total number of tardy transactions with total number of operations is 200, 400, 600, 800, and 1000. Figure 3 shows the penalty (tardy transactions) versus the number of initiated operations. The figure shows that IReIDe outperforms all other approaches. This is due to that IReIDe concerns from the beginning with preventing the tardy transaction and this small ratio occurred locally by the effect of highly throughput.

Finally, to make a comparative evaluation against JB-ML protocol [22] which uses communication cost to modify More-Less approach to maintain the consistency and scalability, we evaluate the performance of IReIDe in term of CPU utilization which is measured under different update workloads (number of update tasks).

Indeed, 2840 update tasks are generated according to normal distribution (11-93) of the worst case execution of update tasks in Table 3 which is depended on number of shared data object in each node.

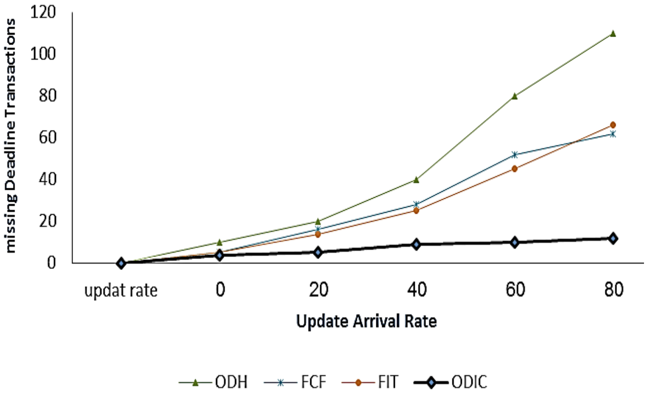

Figure 3. Comparative evaluation from IReIDe and others.

Figure 4 shows the CPU utilization from all nodes in the worst case of each by applying the IReIDe compared with the results of JB-ML protocol in [22] that already outperforms its related works. The results show outperformance of the IReIDe especially in case of larger tasks. IReIDe can reduce the CPU utilization by 17%. The reduction of CPU utilization reaches 20% when the number of queries tasks is more than 270. This reduction may be occurred due to the absence of distributed queries, the lower time which is needed to update only the necessary objects, and the separation between the actual database and replicated data.

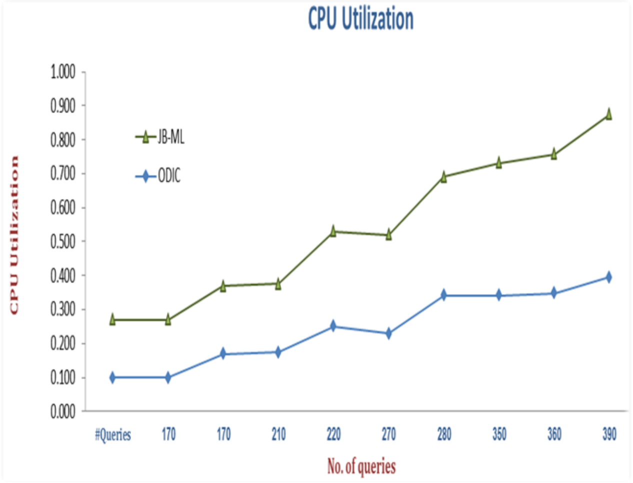

Figure 4. CPU utilization of IReIDe.

5. Conclusions

This work introduces IReIDe as a novel protocol to support the replication for DeeDS and to help in solving the scalability problem using a new clustering technique. The introduced protocol for distributed real-time database acts mainly to map between the network communication time cost and the timing properties of the distributed data. The results show that the IReIDe is able to generate a number of clusters from the distributed database system network sites and reduce the communication overhead between database sites. As consequence, this reduction enhances the performance and increases the chance that DRTDBS can meet critical time-requirements. Also, reducing the large number of network sites into many clusters with smaller number of sites will effectively decrease the replication degree, reduce the consumed system resources and maintain the scalability. These result in better meeting of time constraints. This work tried to maintain the scalability problem of DeeDS which results from full replication strategy to avoid the network delay effect. IReIDe make DeeDS more scalable by managing resources using clustering technique.

Another problem in DeeDS is temporal inconsistency problem which resulted from local commit strategy without coordinating with other nodes. We plan to extend IReIDe by adding a new mechanism to ensure that replicated database continuously converges towards a globally consistent state, where conflicts omitted as possibly at update level.

References

[1] Adelberg B., Molina H., and Kao B., "Applying Update Streams in a Soft Real-Time Database System," *in Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Jose, pp. 245-256, 1995.

[2] Andler S., Hansson J., Eriksson J., Mellin J., Berndtsson M., and Eftring B., "DeeDS Towards A Distributed and Active Real-Time Database System," *SIGMOD Record*, vol. 25, no. 1, pp. 38-51, 1996.

[3] Aslinger A. and Son S., " Efficient Replication Control in Distributed Real-Time Databases," *in Proceedings of the 3rd ACS/IEEE International Conference on Computer Systems and Applications*, Cairo, pp. 34, 2005.

[4] Chen T., Bahsoon R., and Tawil A., "Scalable Service-Oriented Replication with Flexible Consistency Guarantee in the Cloud," *Information Sciences*, vol. 264, pp. 349-370, 2014.

[5] Galeana D., Pacheco H., and Magadan A., "Analysis of Clustering Algorithms for Image Segmentation and Numerical Databases," *in Proceedings of the Electronics, Robotics and Automotive Mechanics Conference*, Morelos, pp. 288-292, 2008.

[6] Hababeh I., "Improving Network Systems Performance by Clustering Distributed Database Sites," *The Journal of Supercomputing*, vol. 59, no. 1, pp. 249-267, 2012.

[7] Hamdi S., Salem M., Bouazizi R., and Bouazizi E., "Management of Real-Time Data in Distributed Real Time DBMS Using Semi-Total Replication Data," *in Proceedings of the International Conference on Computer Systems and Applications,* Ifrane, pp. 1-4, 2013.

[8] Hauglid H., Ryeng N., and Norvag K., "DYFRAM: Dynamic Fragmentation and Replica Management in Distributed Database Systems," *Distrib Parallel Databases*, vol. 28, no. 2-3, pp. 157-185, 2010.

[9] Jaing X., Li J., Xi H., and Hongsheng X., "Distributed Algorithms for a Replication Problem of Popular Network Data," *Journal of Network and Systems Management*, vol. 24, no. 1, pp. 34-56, 2014.

[10] Jannu S. and Jana P., "Energy Efficient Grid Based Clustering and Routing Algorithms for Wireless Sensor Networks," *in Proceeding of the 4th International Conference on Communication Systems and Network Technologies*, Bhopal, pp. 63-68, 2014.

[11] Laarabi M., Boulmakoul A., Sacile R., and Garbolino E., "A Scalable Communication Middleware for Real-Time Data Collection of Dangerous Goods Vehicle Activities," *Transportation Research Part C: Emerging Technologies*, vol. 48, pp. 404-417, 2014.

[12] Lin W. and Veeravalli B., "A Dynamic Object Allocation and Replication Algorithm for Distributed Systems with Centralized Control," *International Journal of Computers and Applications*, vol. 28, no. 1, pp. 26-34, 2006.

[13] Malliaros F. and Vazirgiannis M., "Clustering and Community Detection in Directed Networks: A Survey," Physics Reports, vol. 533, no. 4, pp. 95-142, 2013.

[14] Mathiason G., Andler S., and Jagszent D., "Virtual Full Replication by Static Segmentation for Multiple Properties of Data Objects," *in Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Replication by Static Segmentation for Multiple Properties of Data Objects*, Sweden, pp. 1-8, 2005.

[15] Mathiason G., Andler S., and Son S., "Virtual Full Replication by Adaptive Segmentation," *in Proceedings of the 13th International Conference on Embedded and Real-Time Computing Systems and Applications,* Daegu, pp. 327-336, 2007.

[16] Rajaretnam K., Rajkumar M., and Venkatesan R., "RPLB: A Replica Placement Algorithm in Data Grid with Load Balancing," *The International Arab Journal of Information Technology*, vol. 13, no. 6, pp. 635-643, 2016.

[17] Said A., Sadeg B., Ayeb B., and Amanton L., "The DLR-ORECOP Real-Time Replication Control Protocol," *in Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation*, Mallorca, pp. 1-8, 2009.

[18] Santos R., Bernardino J., and Vieira M., "Leveraging Availability and Performance for Distributed Real Time Data Warehouse," *in Proceedings of the IEEE 36th Annual Computer*

Software and Applications Conference, Izmir, pp. 654-659, 2012.

[19] Sultan T., El-Bakry H., and Hameed H., "Design of Efficient Dynamic Replica Control Algorithm for Periodic/Aperiodic Transactions in Distributed Real-Time Databases," *International Journal of Computer Science Issues*, vol. 9, no. 2, pp. 72-80, 2012.

[20] Sun Q., Qiu Y., Shao Y., and Yan W., "Implementation of Massive Real-Time Database System Using Network Sensors and Sector Operation," *Sensors and Transducers*, vol. 174, no. 7, pp. 123-128, 2014.

[21] Tiwari S., Sharma A., and Swaroop V., "Distributed Real Time Replicated Database: Concept And Design," *International Journal of Engineering Science and Technology*, vol. 3, no. 6, pp. 4839-4848, 2011.

[22] Wang J., Han S., Lam K., and Mok A., "Maintaining Data Temporal Consistency in Distributed Real-Time Systems," *Real-Time System*, vol. 48, no. 4, pp. 387-429, 2012.

[23] Wang W. and Fan S., "Application of Data Mining Technique in Customer Segmentation of Shipping Enterprises," *in Proceedings of the 2nd International Workshop on Database Technology and Applications*, Wuhan, pp.1-4, 2010.

[24] Xu Ch., Sharaf M., Zhou X., and Zhou A., "Quality-Aware Schedulers for Weak Consistency Key-Value Data Stores," *Distrib Parallel Databases*, vol. 32, no. 4, pp. 535-581, 2014.

[25] Zaki M. and Meira W., *Data Mining and Analysis: Fundamental Concepts and Algorithms*, Cambridge University Press, 2014.

Rashed Salem He received his PhD degree in computer science from the University of Lyon 2, France in 2012. He has been a member of Complex Data Warehousing and On-Line Analysis research group within the ERIC laboratory. He is a lecturer at Faculty of Computers and Information, Menoufia University, Egypt. His current research interests mainly relate to database, business intelligence (BI), data warehousing and Big Data.

Safa'a Saleh Currently, she is a Lecturer in Information systems department, Taibah University. She received B.Sc. from Alexandria University. She awarded High Diploma, M.Sc. (by research) in information systems -College of Computing and Information Technology, Arab Academy for Science, Alexandria, 2005 and 2008 respectively. Ph.D. degree in Information System, Menoufia University, Egypt. She has contributed papers in the areas of Data mining, Distributed DB applications and bioinformatics.

Hattem Abdul-Kader He obtained his B.S. and M.SC. (by research) in Electrical Engineering from the Alexandria University, Faculty of Engineering, Egypt in 1990 and 1995 respectively. He obtained his Ph.D. degree in Electrical Engineering also from Alexandria University, Faculty of Engineering, and Egypt in 2001 specializing in neural networks and applications. He is currently a Lecturer in Information systems department, Faculty of Computers and Information, Menoufia University, Egypt since 2004. He has contributed more than 30+ technical papers in areas of Neural networks, DB applications,Information security and Internet applications.