DATA SCIENCE JOURNAL

Salem, R et al 2016 Scalable Data-Oriented Replication with Flexible Consistency in Real-Time Data Systems. *Data Science Journal,* 15: 4, pp. 1-15, DOI: http://dx.doi.org/10.5334/dsj-2016-004

RESEARCH PAPER

Scalable Data-Oriented Replication with Flexible Consistency in Real-Time Data Systems

Rashed Salem[1], Safa'a S. Saleh[2] and H. M. Abdul-kader[1]

[1] Information Systems Department, Menoufia University, Egypt
rashed.Salim@ci.menofia.edu.eg

[2] Information System Department, Taibah University, KSA

Corresponding author: Rashed Salem

Scalability is an increasingly important target for distributed real-time databases. Replication is widely applied to improve the scalability and availability of data. With full replication, database systems cannot scale well, since all updates must be replicated to all nodes, whether or not they are needed there. With virtual Full Replication, all nodes have an image of a fully replicated database and the system manages the knowledge of what is needed for each node to adapt to the actual needs, so that the system can be more scalable. This work proposes a scalable and consistent replication protocol using an adaptive clustering technique that dynamically detects the new data requirements. Because time is critical in such systems, the clustering technique must take into account both the communication time cost and the timing properties of the data. The proposed protocol also proposes a new updated method for addressing the temporal inconsistency problem by skipping unnecessary operations. It allows many database nodes to update their data concurrently, without any need for distributed synchronization. It uses state-transfer propagation with on-demand integration techniques to reduce the temporal inconsistency. The experimental results show the ability of the proposed protocol to reduce the system resources consumed and improves system scalability while maintaining consistency.

Keywords: clustering; distributed systems; real-time databases; replication

1 Introduction

A real-time database system (RTDBS) is a time constraint system in which a transaction requires servicing on or before its deadline (Aslinger & Sang 2005). The resulting value from missing the 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 which has a fatal effect on the system, missing a firm deadline gives no value, while missing a soft deadline may leave some value in the computation for some time (Wang, Li-Wei & Yang 2011). Recently, the demand for real-time databases is increasing, many applications such as e-commerce, mobile communications, accounting, information services, medical monitoring, nuclear reactor control, traffic control systems, and telecommunications require real-time data support (Hamdi, Salem & Bouazizi 2013).

The long time that may be taken to obtain data from a remote site can make temporal data invalid, causing the transaction deadline to be missed. Replication is a known technique to improve the availability of avoiding late transactions (transactions that miss their deadline) (Farn, Li-Wei & Ya-Lan 2011). At the same time, predictability is one of the most important features of RTDBS and is often more important than consistency, which can be relaxed to improve the predictability of data access. This is due to the fatal effect of missing a deadline in a hard real-time system and the reduced service of soft real-time systems. For real-time systems, predictable resource usage is the most essential design concern to enable timeliness. This needs detailed prior knowledge about the worst case execution order of concurrent transactions, where the highest resource usage occurs (Hamdi, Salem & Bouazizi 2013). Distributed processing is the main source of unpredictability so the need for full replication is seen as an urgent solution for such a problem (Andler et al. 2007).

However, full replication makes all data objects available at each local node for any transaction giving the possibility of achieving good predictability. It suffers however from a scalability problem as it consumes system resources because the system must replicate all updates to all of the nodes, regardless of whether data will be used there or not. Also, updates in fully replicated databases must be integrated at all nodes, requiring integration processing of updates at all nodes (Mathiason, Andler & Son 2007).

Virtual full replication is introduced mainly to improve scalability, especially in large systems. With it, every node carries a full image of the database, and the updates will be replicated where they are needed only. This creates a perception of full replication for the database user, such that the database can be used in the same way as is possible with full replication. On the other hand, the degree of replication becomes lower than that in a fully replicated database according to the actual need. This can support scalability as the resource usage of bandwidth, storage, and processing which are wasted in a fully replicated database system will be reduced with virtual full replication. In fact, preserving system resources can enable DRTDBS to better meet time requirements while keeping the ability of the system to scale (Andler et al. 2007). At this point, some knowledge about the access pattern is required to discover the data needs of each node. In our work, we use a dynamic clustering technique to track the change of data needs at each node.

Clustering of database nodes is a vital issue in parallel and distributed database systems which can help them to face the challenges of meeting time requirements. It is used to reduce the communication cost between distributed database nodes. Clustering can be seen as a method of grouping network sites according to a certain criterion to increase the system performance. However, reducing the large number of network sites into many clusters with a smaller number of sites will effectively decrease the response time, resulting in a better capability to meet time constraints. It can be considered as an approach for both limiting the degree of replication, and achieving a virtually fully replicated database. The degree of replication is a result of allocating nodes to the clusters where its data objects are accessed. This requires typically much fewer nodes than used in a fully replicated database (Hababeh 2012). Because time is critical in such systems, the clustering method must take into account the timing properties of both the communication network and data of the access pattern. The aim here is to improve availability and to enable DRTDBS to meet critical timing requirements (Jon, Norvald & Kjetil 2010) while improving the ability of the system to scale.

In general, optimistic protocols are suitable for DRTDBS replication. They allow reading or writing operations on data without any prior synchronization, permitting propagation in the background and resolution of conflicts after they occur. At the same time, pessimistic protocols are not suitable at all for DRTDBS because they allow synchronization on replication, with blocking of other operations during update. This may badly affect the meeting of time requirements (Laarabi et al. 2014).

The proposed protocol also depends on detached replication, which is a form of optimistic replication (Gustavsson & Andler 2005). Propagation of transaction updates to remote nodes is delayed until after the transaction commits. This relaxes ACID (Atomicity, Consistency, Isolation, Durability) properties (Harder & Reuter 1983) and leads to a degree of temporal data inconsistency which the application must be tolerant to.

In fact, DRTDBS is allowed to be weakly consistent temporarily at the expense of predictability because in DRTDBS, the local availability and efficiency are more critical than immediate global consistency (Laarabi et al. 2014). There is a tradeoff between the global consistency and the local deadline with predictability. To improve the consistency, the proposed protocol uses state transfer propagation instead of operation transfer. This acts to decrease the inconsistency duration as it needs only to record and transmit the final value of the data object. On-demand updating is used in general to avoid unnecessary updating operations which affect the system performance and the ability of RTDBS to tolerate the missing of time constraints. It links the execution of the update operation with special criteria every time data is requested (Saito & Shapiro 2005).

The contribution of this work is to propose a new replication protocol based on the concept of virtual full replication with state-transfer propagation and on-demand updating to address the scalability and consistency problems. The proposed protocol uses a clustering technique to define the optimal number of clusters of nodes while maintaining consistency by using an on-demand integration technique. Finding the optimal solution that groups distributed RTDBS nodes into logical clusters minimizes the consumption of system resources and increases the transaction response time while allowing for parallel replication that improves the consistency and yield the possibility of improved scalability without issues of performance (Jon, Norvald & Kjetil 2010). The proposed protocol also allows many database nodes to update their data values concurrently, without any need for distributed synchronization. Using the state-transfer technique with an on-demand integration process reduces the temporal inconsistency which is caused by the allowed detached updates. In this way, RTDBS can avoid unnecessary update operations and the resulting workload of scheduling and conflict resolution. This enhances the system performance, increases the chances of meeting the time requirements, and improves the consistency of DRTDB. This work also introduces the concept of on-demand integration, in which integration with new values of a data object is linked to its request. The idea is based on finding and transmitting the final values of objects, instead of the sequence of operations, which will decrease the temporal inconsistency duration by skipping the execution time of the transferred update and avoiding the need for extra control of scheduling and conflict management. This forces some changes in the order of the integration, propagation, and execution phases of replication. The proposed protocol is a completely decentralized system, i.e. there is no coordinator or even control node. The decision regarding propagation and integration is taken without any synchronization between sites. Two aspects are important in the proposed protocol; the first is the combination of clustering and dynamic adaption of replication as a unified process, and the second is linking the integration process with the demand of the data object.

The remainder of this paper is organized as follows: Section 2 discusses the previous work which is related to the present work. Section 3 presents the proposed protocol approach for scalable and consistent replication in Real-time databases Section 4 covers the experimental study to evaluate the introduced algorithm. Finally, the conclusion is provided in Section 5.

2 Related Work

Using the term of scalability, there are many works in different areas aimed at minimizing the communication cost such as (Lin & Veeravalli 2006; Hababeh 2011).

Although there are many researchers working in the field of DRTDBS replication such as: (Aslinger & Sang 2005); (Li-Wei & Yang 2011); (Hamdi, Salem & Bouazizi 2013); and Said, Sadeg & Ayeb 2009), few of them address the concept of virtual full replication such as described in (Mathiason, Andler & Son 2007) and (Gustafson et al. 2004). The work by (Mathiason, Andler & Son 2007) tries to maintain the scalability at execution time and to improve the access time to data at any node by grouping data objects into segments. This is called ViFuR-A as it was aimed at adapting replication over time to actual data needs of database clients. But ViFuR-A ignores the use of communication properties during the segmentation phase. Also, it uses a fixed replication degree for each segment, which often does not occur. The replication degree is a dynamic concept that is based on the requirement of each node.

By thinking of clustering the database nodes, work such as those of (Srikanth & Prasanta 2014) and (Ivanova, Kersten & Nes 2008) are found. However the work by Hababeh (2012) must come to the fore. This work introduces an intelligent clustering technique that segments the distributed database network sites into disjoint clusters according to the lowest average communication cost between network sites. This work has succeeded in reducing the communication cost which we believe can improve the predictability of DRTDB. However we need to extend their clustering algorithm to include the timing features of RTDB in the clustering criteria in addition to the network properties in order to obtain better predictability. This can reduce the communication traffic and improve the performance of DRTDB.

On the other hand, consistency is the main issue when thinking of replicated data in distributed systems (Meixing, Sushil & Krishna 2014). Maintaining the temporal consistency of time-constraint data is an ultimate goal of most researchers who are working in DRTDBS due to the relaxation of consistency at the expense of predictability. Many solutions with various strategies have been introduced to address the inconsistency problem. Some of these are concerned with managing the scheduling operation to improve the consistency such as in (Golab, Johnson & Shkapenyuk 2009); (Hamdi, Salem & Bouazizi 2013); and (Song et al. 2012). We believe that scheduling, conflict resolution, and other operations which are related to the update execution can be skipped at the expense of saving time in order to improve the chance of increasing consistency while maintaining the freshness of the data where needed.

Others works target minimization of the execution duration of the updating operation including work by (Aslinger & Sang 2005); (Said, Sadeg & Ayeb 2009); (Jiantao et al. 2012); and (Xiong et al.2010). The idea of reducing the execution time of this category of work is close to our vision. It is a good idea to reduce the inconsistency duration by reducing the execution duration of the updating operation. However, it would be better to skip this execution time and apply on-demand integration.

By exploring the works in on-demand updating, the work by (Gustavsson & Andler 2005) comes to the fore. This work applied the on-demand concept to reduce the number of unnecessary updates, and thus minimizes the CPU utilization for processing updates. They cannot guarantee temporal consistency by changing the period of update job generations. Also, the work by Chen et al. (2013) is another example of the on-demand selective approach. They introduce a freshness/tardiness (FIT) algorithm as a novel mechanism for maintaining consistency and scalability in massive distributed data. This mechanism highlights the importance of using the communication cost with state-transfer updating. In the state-transfer update, FIT basically relies on installing or skipping pending updates. In the operation-transfer model, FIT efficiently determines the optimal number of updates to achieve maximum benefit. The idea of depending on the role of the communication cost of Jiantao et al. (2012) makes it most closely related to the present work, but we still believe in using state-transfer instead of operation transfer to save the execution time without any need for extra scheduling or conflict control. This will give a greater chance of improving the consistency of the DRTDB model by reducing the inconsistency period.

The proposed replication protocol is also similar to the replication approach in DYFRAM (Jon, Norvald & Kjetil 2010) in several aspects: 1) DYFRAM, like the proposed protocol, uses the access pattern to allocate the data fragments dynamically. In general, working dynamically is an important aspect toward fully automatic tuning. 2) DYFRAM is a decentralized system using an adaptive data placement. The key difference between DYFRAM and the proposed protocol is that the proposed protocol is based on the mapping between the timing property of RTDB and the communication cost between the distributed nodes.

3 The Proposed Approach

This section presents the architecture of the proposed protocol approach that supports the scalability and consistency of distributed real-time database systems. However, the problem of scalability in eventually consistent distributed real-time databases can be applicable to any database system. In this section, we propose a generic architecture for such database systems, with the key components needed to implement virtual full replication with state-transfer propagation and on-demand integration. Each database node in such architecture is a peer node, and the key functional components that are used by each node are presented in **Figure 1**.

The *Cluster Manager component* is responsible for supporting virtual full replication via identifying and managing the list of target nodes that must receive the replicas of data objects. Its function is to discover the shared nodes according to the access pattern and use this information to cluster the distributed database nodes virtually based on the communication time cost of the physical network and the time properties of the transmitted data. The *Cluster Manager* is also responsible for establishing the clusters centrally and managing them in a distributed manner. The resulting clusters at the local node are stored in the *subscribers* datastore, which is adapted by adding and/or removing nodes to/from the cluster according to any change in the pattern access.

The *Replication component* (referred to as the *Propagator* due to the importance of the propagation process) is responsible for implementing virtual full replication by managing the operations of distributing and receiving replicas. This component is responsible for distributing the updated values to shared nodes. Also, it stores the received values in the *Replication directory* (*Rep_Dir*) and participates with other components in adapting the clusters (*subscribers*) by adding new nodes or removing passive nodes.

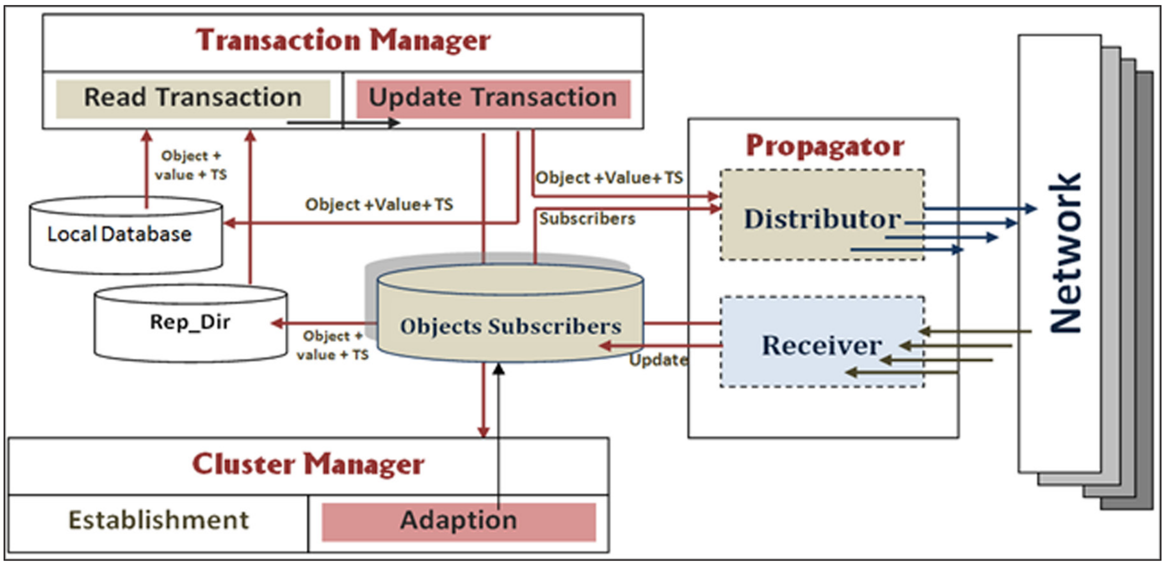

Figure 1: System Node Components.

The *Replication Directory* (*Rep_Dir*) is an important datastore that plays an important role in improving the consistency. It holds the received information from the updating node (sender) and provides the running transaction with suitable data during the integration phase. *Rep_Dir* is updated dynamically with each received message by adding new values and deleting a number of old entries of the same object at the same node based on the time stamp, or updating the current entry.

Subscribers is a Boolean two-dimensional data structure that identifies the shared nodes (*subscribers*) of each data object which are together in same cluster. It is used by the propagation process to distribute the updates of an object to all listed nodes for the individual object.

The *transaction manager* is responsible for integration with the newest replicated value. This is linked with any read or write operation on any data object. It returns two copies of the object value, one from the local data storage and the second being the most recent replica from *Rep_Dir*. The integration phase starts by comparing the time stamp of each copy. By the end of the transaction, the local value data object is updated using the more recent value which results from the write transaction or from the comparison operation if the local value is older and the transaction type is read.

3.1 The cluster component

To make a decision regarding the optimal number of clusters, some knowledge from the past about the accessed data objects is needed. This helps in discovering the access pattern which can be adapted later by each node (Jiawei & Micheline 2006; Kumar et al. 2007). After discovering the access pattern, the clusters are generated based on the rule that the communication cost between two sites in the discovered pattern is less than the validity of the replicated data; i.e. each object must be replicated within its validation period (we refer to such data as 'allowed data').

The produced clusters are allowed to overlap but they are prevented from intercommunication with each other. Each database node contains a copy of the *subscribers* datastore to define the set of shared nodes for each data object which form a cluster. The *propagator* in each node uses the *subscribers* to replicate any update on the data object to the selected nodes that define the individual data object. The replication operation will execute in *O(n)*, where *n* is the number of *subscribers* for each data object cluster. The following list describes the parameters of our proposed algorithm:

- Communication Cost matrix ***CC*(S_i, S_j)**: the cost of transmitting the data object in ms/byte between any two nodes S_i and S_j in the distributed system.
- Log matrix **(*OT*)**: working set (objects used by the transaction) for all transactions at each node *S*(*i*).
- Deadline (*O*)(*N*): the minimum deadline or time-constraint of the transaction on each node.
- *Subscribers* (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).

The task starts by scanning transactions in the log matrix (OT) to extract the minimum value of the deadline (*DL*) of each transaction, or the *Validity Duration* (*VD*) of the exploited data, to produce a minimum deadline matrix. This matrix holds information about the access pattern of each object. Then, it identifies the allowed nodes; those nodes have a time cost with the current node which is less than the minimum deadline of the shared data object. In this way, this module initiates the cluster by discovering the shared nodes of each object in the access pattern of an individual node. **Algorithm 1** outlines the main steps of this module.

In general, a static database configuration is very limiting as it is based on an object access which can be changed during work. With direct on-line adaptation of clusters, the *subscribers* matrix (objects × nodes) is updated at each node separately according to the change of data needs at each node. Typical adaptation occurs when any node requests a data object and this node is not specified as a member in the data object cluster. This triggers a set of tasks to add this node to that cluster which may then need to remove the most passive nodes from that cluster to preserve the threshold of replication degree.

Each node contains an *Active_list* data structure which holds the most recent node identifiers that access each of the data objects which are sorted according to the time stamp of activity. The most active node is the node that has the most recent access to the object. Receiving each update message results in the updating of the *Active_list* to modify the order of the most recent node of each object based on time stamp. Adding new nodes starts when the node finishes any updating transaction then tries to send the resulting updates to the object's *subscribers*, and discovers that it is not a *subscriber* of the current object. It sends a broadcast message with a request to add it as a *subscriber* of the object. Other nodes which are *subscribers* of that object use the *Active_list*

Algorithm 1: Identifying Object *subscribers.*

Inputs → Log matrix(*OT*), total number of nodes (*N*), total number of objects (O), *CC*[*N*][*N*]

```
For each row ∈ OT do using counter i:
  deadline[OT.objectID][OT.Node]= min (VD[i],DL[i])
loop
Set Boolean matrix:Subscribers [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
          Subscribers[temp[x]][ObjectID][temp[y]] ← True;
          Subscribers[temp[y]][ObjectID][temp[x]] ← True;
        End if
      loop
    loop
loop
```

Outputs → *Subscribers*[N][O][N]

to make a decision regarding removal of the least active node when the number of *subscribers* increases to a specified threshold (70% in our case) of the total number of nodes, at which point new nodes may be added to the object *subscribers*. Removing a node from a cluster is linked with adding a new node to cluster. Removing nodes from the existing cluster depends on the change of the object access pattern. Passive nodes of any object are those that do not access the specified object for a period of time while there is recent access by other nodes.

3.2 The replication component

The trade-off between data consistency and meeting the time requirements in DRTDBS represents the ultimate challenge of any work in the field of real-time data replication. The replication component of the present work proposes a novel, tolerant, optimistic mechanism for distributed real-time data to improve consistency while meeting time requirements and allowing the system to be more scalable. After reducing the consumed resources and decreasing the replication degree, the next question is how to reduce the time consumed by the replication process, or in other words, how to decrease the duration of inconsistency? To achieve this, the propagation phase is based on state-transfer propagation. In this context, we aim to reduce the duration of inconsistency by omitting the time of execution of the updated replica. Note that distributing the replica to only those nodes that need them will decrease the consumed communication cost and reduce the traffic load.

Reducing the network traffic load will consequently reduce the overhead in the communication link, which gives an opportunity to accelerate the transmission rate (Yan, Tilman & Weibo 2011). Also, transferring the final state of the updated object instead of the whole update transaction, especially in the most usual case of small data updates, will reduce the duration of the inconsistency by avoiding the time of transaction execution and improving the system performance.

The propagation process has two phases: the distribution phase and the receiving phase. When a transaction that accesses an object for a write operation has committed, the *distributor* is called. The latter starts with checking the *subscribers* datastore to extract the shared nodes of the updated data object. Then it uses the extracted list to propagate the update to the specified *subscribers*.

If the current node is not listed as a *subscriber* for the current data object, the *distributor* will send a broadcast message with a request to update the *subscribers*. The message contains the current *object identifier* (O), the updated *value* (Val), and the current *node identifier* (N) in addition to the time stamp (TS). Such values are used by the receiving nodes to add the received *node identifier* to its *subscribers* datastore for the received *object identifier*. **Figure 2** summarizes the procedure of the distribution phase. Also, the receiving node stores the received object *value* to update the object *value* and the corresponding time stamp in the *Rep_Dir* (replication directory) datastore.

The node which receives the propagated message will call the receiver module that executes the *B_receiving* method in case of a broadcast message or executes the *R_receiving* method in case of regular update message. Receiving a broadcast message means that there is a new node of the object cluster. The *B_receiving* method checks whether the current node is a *subscriber* of the specified data object to add the received node ID as a new *subscriber* of the specified object. Also, it updates the *Active_list* datastore which holds the most recent node identifications for each object. The *B_receiving* method also uses the received information to update the *Rep_Dir*. In the case of a regular received message, the *R_receiving* method only updates the *Rep_Dir* datastore using the received data and refreshes the *Active_list* datastore to modify the order of the recent active nodes of the object.

Note that all nodes of a given replica (updating or receiving node) work independently. The sender node that updates the data object of a specified replica will: (1) create a new replica of the updated object, and distribute it to the *subscribers* in a cluster of the specified replica, and (2) add itself to a cluster of the specified replica if it is not found. This may require to modification of the entire list in the cluster. The receiving node will: (1) update the needed datastores (*Rep_Dir*), (2) add the sender node to a cluster of the specified replica if it is not found, which may cause deletion of existing *subscribers* to preserve the threshold of replication degree, and (3) use the *Rep_Dir* later to integrate the data.

3.3 The update model

The optimistic replication algorithm allows updates to be executed on a single node at a rate that is independent of propagation operations. It also allows integration with the propagated messages (which are received from remote transactions) to occur locally at a receiving node according to a local scheduling policy and a local conflict detection.

Using state-transfer propagation with on-demand integration aims at: (1) reducing the transmission time by replicating data to only a subset of nodes, (2) reducing the inconsistency duration by omitting the time of execution of the updated replica. Note that, transferring the final state of the object instead of the whole update transaction will reduce the inconsistency duration by avoiding the time of transaction execution and improve the system performance. The update model of the proposed protocol is based on optimistic replication and benefits from the continuous convergence (CC) protocol introduced by (Gustavsson & Andler 2005). The CC protocol is designed to meet three important database requirements: local consistency, local predictability, and eventual global consistency. The safety critical transactions in a time-constraint database must have a predictable and sufficiently short execution time in a locally consistent state, in other words, they achieve local predictability and local consistency. Eventual global consistency allows the systems to be consistent when they become quiescent (i.e. should all transaction activity cease) (Terry et al. 1995; Shapiro et al. 2004). The proposed protocol achieves the requirement of predictability, local, and eventual consistency via propagating the resulting states from the updates of a subset of data continuously to a subset of

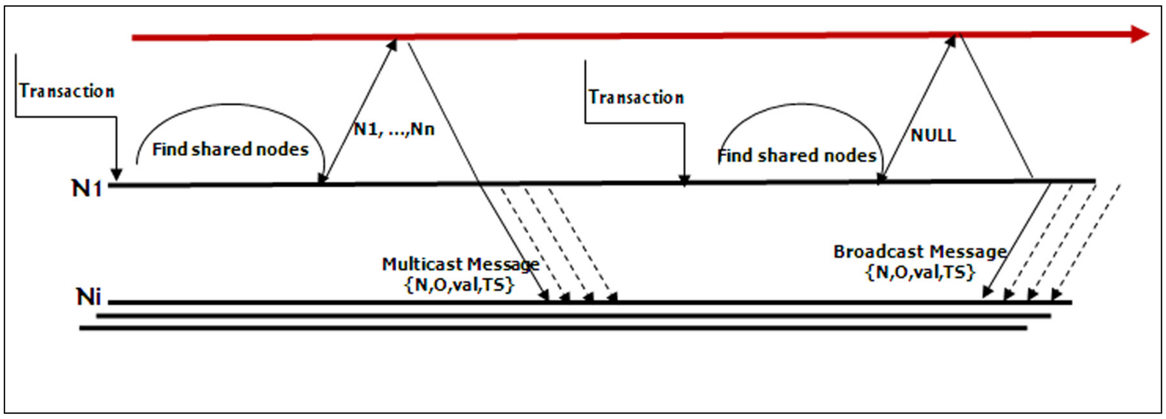

Figure 2: Distributor (publisher) module.

Algorithm 2: Rep-read(O): Retrieving recent information.

Inputs → data object identifier ObjectID

```
Set O ← Object_ID
Set i ← 0
Temp[][] ← Ø
For each row ∈ Rep_dir[][Object_ID] do
  If Rep_dir[row][ ObjectID]= O then
       Temp[i][value] = Rep_dir[row][ value]
       Temp[i][TS] = Rep_dir[row][TS]
     Break;
End if
i ← i + 1
Loop
  sort(temp) on TS descending
Object_recent[value] ← Temp[i][value]
Object_recent[TS] ← Temp[i][TS]
```

outputs → Temp[i][TS] & Temp[i][value] of most recent value of ObjectID

Algorithm 3: On-demand Read Transactions.

Inputs → Object Id (O)

```
    //Read object's value and timestamp from local database
Set SrcState ← Object_value
Set SrcTS ← Object_ TS
    //Read object's value and timestamp from Rep_dir in
Set RState ← Rep_read.get(value)
Set RTS ← Rep_read.get(TS)
    IF RTS<SrcTSthen // less recent
    State ← RStateelse State
    Else State ← SrcState
    End IF
    //Update local value of to Object_Id V
    Local_Database [Object_Id][value] ← State
    Return(State)
```

Outputs → value for (O)

needed nodes only and as soon as possible. Conflicts are continuously and optimistically resolved locally by forward conflict resolution.

On the receiving node, the integration phase in the proposed protocol is also done locally but it is linked with a request of the data object using two central datastores which are: the local database and the replication directory (*Rep_Dir*). The received replica which holds the newest value of the updated data objects is stored to the *Rep_Dir* datastore after deletion of the older versions of the object's replica from the same node according to the received time stamp of the replica. The request of the data object by reading or writing enhances the updating of the local value with the most recent global value. In other words, every time a data object is requested, the on-demand updating technique uses the most recent value of the specific data object to update the local value. This optimistic operation ensures that the updating occurs only if it can fit the consistency state. If the transaction needs to obtain any value of any object, it returns the value and time stamp of the newest record of the specified data object from *Rep_Dir*, and the same data from the local database also. The value returned from *Rep_Dir* and the corresponding values from the local database are compared in order to use the more recent version to support consistency. The transaction processing phase uses the *Rep-read* method when it needs to obtain any value of any object. This method returns the value and time stamp of the most recent record of the specified data object. The algorithm for this method is presented in **Algorithm 2**.

According to the load hypothesis (Burns & Wellings 2001), a hard real-time system can only handle a limited rate of load due to time constraints. Since integration is also done by transactions, the integration phase must also be taken into account when determining the transaction load for a node. Within a given time period, a fixed number of transactions are specified in order to ensure that deadlines are met. To achieve better performance, the number of messages of a node can be limited in terms of the number of local integration transactions per time period due to the load hypothesis on real-time databases per node. So, the maximum number of arriving transactions per time period must be bounded. It is assumed that a transaction is only allowed to access a limited number of objects.

The transaction uses the values returned from the *Rep-read* method and compares them with the corresponding values from the local data in order to use the more recent version to support consistency. **Algorithm 3** presents the read transaction that is responsible for updating (integration) the local value of the specified data object if the global version is more recent. The write transaction does the same as it calls the read transaction, in addition, it will update the local value with the new value while calling the propagation module to distribute the new value.

4 Experiments and Results

The proposed protocol is implemented using a real-time database which is created using the SQL server 2012 database. It is implemented using a completely connected network consisting of 10 nodes supported by a fully-distributed database over different areas. For simplicity, each data object contains only one time-constraint attribute. **Table 1** summarizes the set of parameters and the baseline settings for the experiment. In fact, the settings of the evaluation experiments are very similar to those of the closest related work in (Mathiason, Andler & Son 2007) and (Hababeh 2012).

To evaluate the scalability and performance of the proposed protocol we depend on a variety of scalability metrics which have been developed for massively distributed computation to evaluate the effectiveness of algorithms from a scalability perspective (Xiang et al. 2013). These metrics are determined as the quantified utilization of: storage; CPU; and bandwidth (communication cost). To investigate the impact of the proposed protocol on the scalability of the system, we use two workloads in each node by initiating a number of updates (write transactions) on a randomly selected set of data objects, one represents a lighter workload while the second represents a heaver workload. Both workloads are expressed by the number operations per second and are presented in **Table 2**.

To evaluate the proposed protocol using the storage requirements metric, **Table 3** presents the quantified storage (in bytes) that are used by clusters (with different capacities) in the cases with and without the proposed protocol i.e. full replication where the storage media is consumed on all nodes for all objects.

Parameter	Default value
# Nodes	10
Database size	100 objects/node
Data size	64 byte
Validity interval	1–2.8 sec
Transaction size	5 update operations
Transaction arrival time	[200–1,000]

Table 1: Experiment settings.

Node	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10	total
Workload 1	21	23	26	22	30	43	40	43	41	30	319
Workload 2	101	103	106	102	110	123	120	123	121	111	1,120

Table 2: Number of write operations generated in each node.

Cluster Capacity	2	3	4	5	6	Full
Storage cost	128	192	256	320	384	**640**

Table 3: Storage Cost Evaluation of the proposed protocol.

Comparing the impact of the present work to the work by (Mathiason, Andler & Son 2007), it is found that they measured the storage need for an increasing number of nodes compared to full replication and discovered that their approach in some cases consumed 12–15% more storage. In contrast to the proposed protocol results, which achieved a very high reduction in storage (around 80%), in spite of the fact that the proposed protocol used the same number of nodes and the same number of data objects with the same size.

In the context of the number of replicas, each node has its own specific number of replicas depending on the replication degree of its shared objects. The maximum number of replicas resulting from the proposed protocol is 123 replicas of 42 objects, as in the case of node 8. For comparison, we consider one case that was used by (Mathiason, Andler & Son 2007), where the accesses within a fixed number of replicas was 300. This case was not the worst case, as they reported. Compared with 123 replicas (the worst case resulting from the proposed protocol) the proposed protocol has a lower storage cost.

Using the consumed bandwidth metric it is assumed that every update of the data object use one network message, so update messages are equal in size. Using the settings of the first workload (**Table 2**), the total communication cost that can be consumed if all replica updates occur is 290 ms/byte from the first workload, while the corresponding value in the case of full replication is 1,479 ms/byte. This value increases with the second workload to 358 ms/byte. In other words, the proposed protocol reduces the consumed bandwidth by around 80% as it lowers the number of replication messages by a reduction in the number of target nodes.

To make a comparative evaluation against the DYFRAM approach (Jon, Norvald & Kjetil 2010), which is one of the most related approaches to the proposed protocol, we use histograms to record the number of migrated objects during the execution of a number of updates with the first workload (**Table 2**) that are distributed over all nodes. Although their work is different from the proposed protocol, as it depends on the fragmentation of data before replication, it is similar to the present work in many respects as described in Section 2. They evaluated their work with four workloads, we only considered the first two workloads to compare with our results because their settings are close to the settings of the present experiment. **Table 4** presents the total number of transmitted messages from all nodes using the proposed protocol compared with the corresponding of DYFRAM. The noticable reduction of transmitted objects by the proposed protocol is due to the reduction of the number of target nodes by identifying the needed nodes of data objects.

In the context of the average communication cost, the DYFRAM approach consumed 75.2 with a maximum value of 136.0 and a minimum value of 14.0, compared with an average value of 14.5 using the proposed protocol (note that the total average of the communication cost obtained by their four workloads was 16.45). They also pointed to a 50% reduction in the communication cost against the case of full replication, compared with an 80% reduction in the communication cost from the proposed protocol.

To make another comparative evaluation against the clustering method by (Hababeh 2012) in terms of the maximum load (bit/sec) which is generated from all nodes as an indicator of the consumed communication cost, we considered case of 10 nodes compared with (Hababeh 2012) and his most related work (Kumar et al. 2007) and (Fronczak et al. 2002). **Figure 3** depicts the load of the clustering methods under comparison.

The transmission reduction is another criteria to evaluate the proposed protocol in comparison with the most related work. To make this comparative evaluation against DYFRAM as a related work to the proposed protocol, the access rate of 3,190 using the proposed protocol compared to 4,000 by using DYFRAM is depicted in **Figure 4**. The transmission reduction by DYFRAM reached 40.5% in the general case and 52.1% in the optimal situation. The transmission is reduced by the proposed protocol from a value of 40% within the largest cluster to 80% within the smallest cluster, with an average of 60%.

	# write operations	**# migration objects**	
		Proposed work	**DYFRAM**
Workload 1	984	620	1,385
Workload 2	4,008	1,081	3,173

Table 4: Number of migration objects (transmitted messages).

To evaluate the performance of the proposed protocol, CPU utilization is recorded using the second update workloads (**Table 2**). This distribution is dependent on the number of shared data objects in each node. **Figure 5** shows the CPU utilization from all nodes in the worst case of each by applying the proposed work compared with the results of the JB-ML protocol in (Jiantao et al. 2012), which latter already outperforms their related work (ML) in (Yan, Tilman & Weibo 2011). The results shows the better CPU performance of the proposed work, especially in case of larger tasks. The proposed work can reduce CPU utilization by 17%. The reduction of CPU utilization reaches 20% when the number of query tasks is more than 270. This reduction may be 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 storage of local database and replicated data.

In fact, the proposed protocol, as an optimistic replication protocol, depends on the eventual consistency model in which the importance of highly consistent data can be relaxed at the expense of availability. The ability of an optimistic replication system to rapidly achieve eventual consistency at the expense of meeting time constraints is a crucial factor for the usefulness of the system. To a great extent, the ability of the system

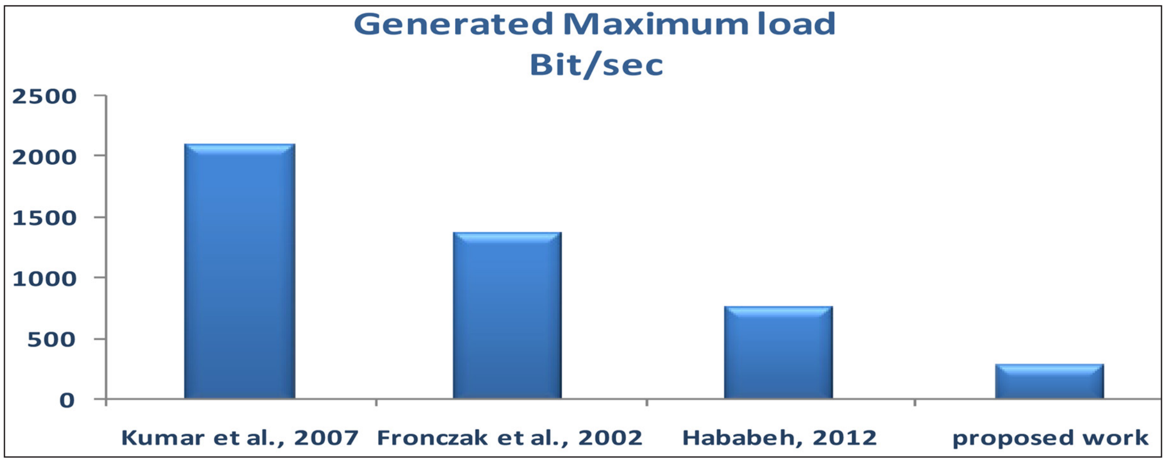

Figure 3: Maximum Load Evaluation.

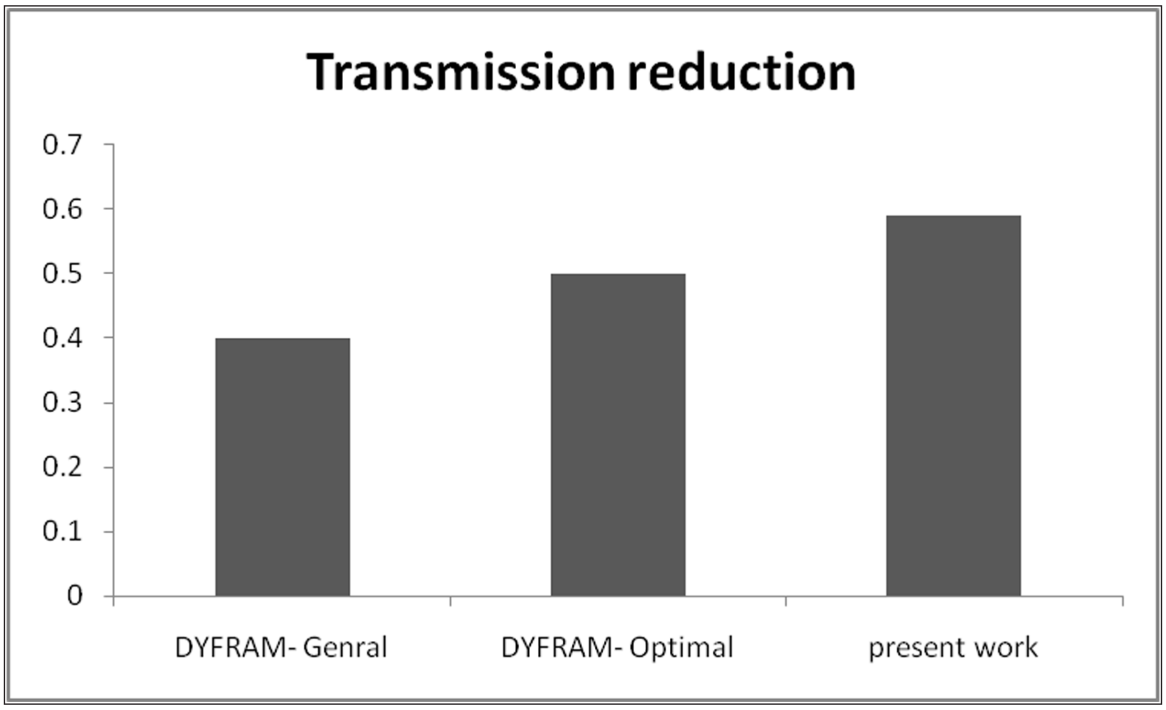

Figure 4: Comparative Evaluation with DYFRAMA.

to resort to a consistent state as fast as possible can be a good indicator of a good eventual consistency. As stated in (Barreto & Ferreira 2010; David, Sherif & Liang 2013), the most common metrics of eventual consistency are: transaction commit ratio (success ratio), or transaction delay.

Chen et al. (2013) evaluate the consistency of their mechanism in comparison with other related works such as ODH (Adelberg, Garcia & Kao 1995) and FCF (Buttazzo, Spuri & Sensini 1995) in terms of the average number of tardy transactions to the offered throughput (operations per second). To make a comparative evaluation against the FIT mechanism and its related works, we initiated a number of random updates (write transactions) on selected data objects in all nodes, and used the performance monitor to record the total number of tardy transactions where the total operations number is 200, 400, 600, 800, and 1,000.

Figure 6 shows the penalty (number of tardy transactions) versus the number of initiated operations. The figure shows that the proposed protocol outperforms all other approaches. This is due to the fact that the proposed protocol is concerned, from the beginning, with preventing tardy transactions and this small ratio occurs locally through the effect of high throughput.

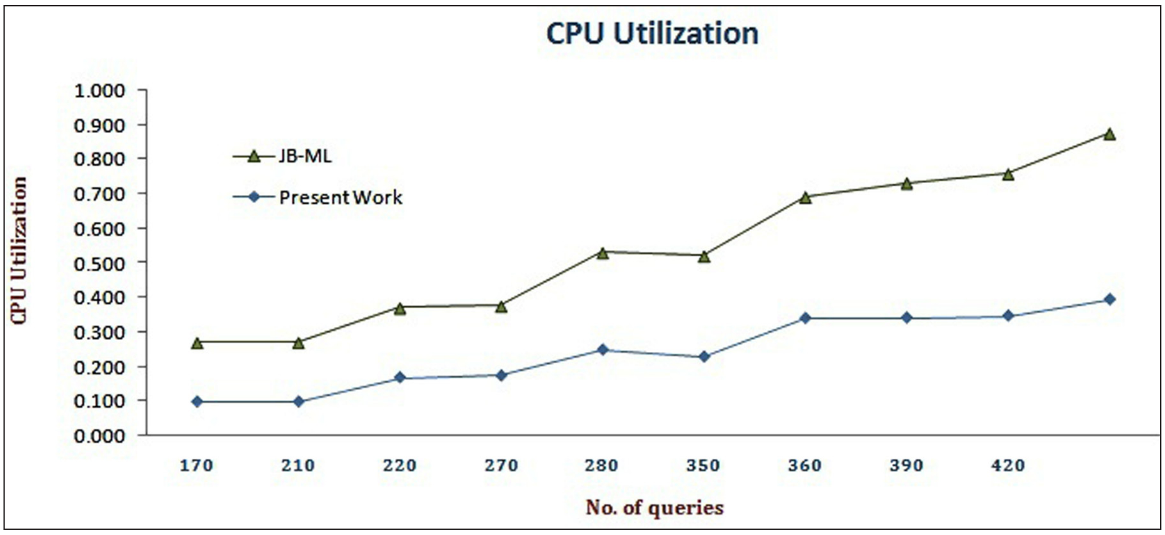

Figure 5: CPU utilization of present work.

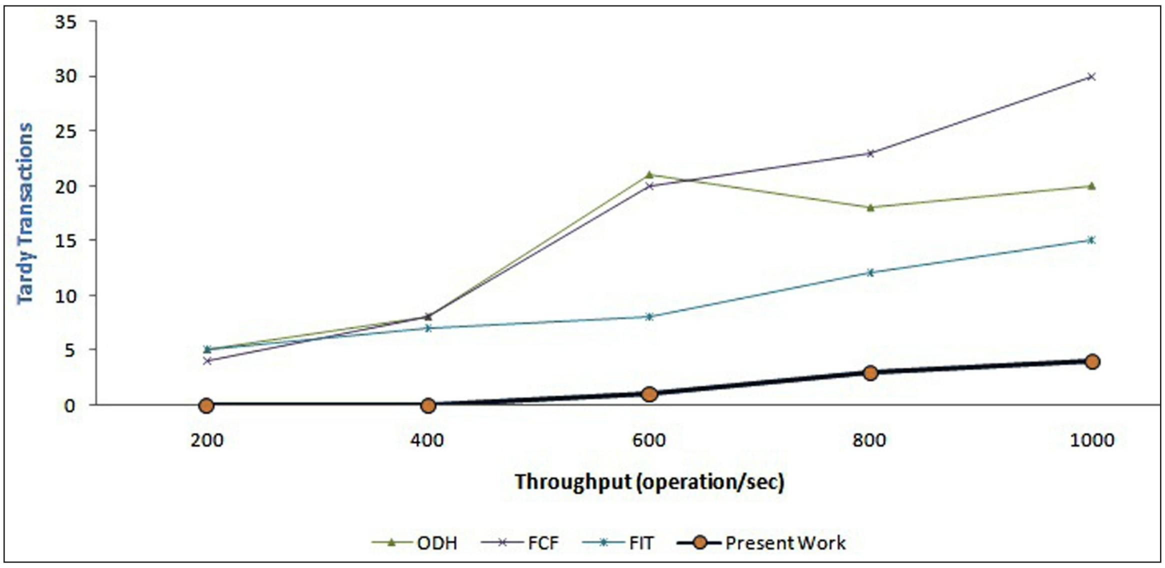

Figure 6: Comparative evaluation from the proposed protocol and other works.

5 Conclusion

This work proposes a novel replication protocol in DRTDBS. To address the scalability problem in such systems, it uses a dynamic decentralized clustering technique to achieve virtual full replication. The clustering technique acts mainly to map between the network communication time cost and the timing properties of the access pattern of the distributed data. Also, the proposed protocol addresses the known consistency problem in DRTDB using state transfer propagation by skipping unnecessary operations. It allows many database nodes to update their data concurrently without any need for distributed synchronization, permitting propagation in the background and resolution of the conflicts after they occur. State-transfer propagation removes the possibility of adding any extra conflict management loads which can be caused by the effects of update-transfer propagation. This saves time, improving the opportunity for meeting the time requirements and reduces the duration of inconsistency which improves consistency. It reduces the consumed resources in term of CPU and memory utilization to yield the opportunity for improved scalability. In this work we introduce the concept of on-demand integration, which links the integration with new replicated values of the data object with its request. It ensures that the replica is used in a validation manner, avoiding any fatal effects in the case of missing the deadline. Results show that the proposed replication protocol is able to enhance the scalability and performance and increase the chance that DRTDBS can meet critical time-requirements. Our analysis of scalability shows that a vast amount of system resources can be saved by considering the actual need for data. The results also show that this replication protocol decreases the temporal inconsistency problem which results from the local commit strategy without coordination with other nodes. For future work we aim for a further reduction in the number of objects for which conflict detection and resolution is necessary. Also, we plan to implement the proposed techniques in real applications of RTDB such as mobile and wireless sensor networks, looking into the scheduling control issue.

Competing Interests

The authors declare that they have no competing interests.

References

Adelberg, B, Garcia-Molina, H and **Kao, B** 1995 Applying update streams in a soft real-time database system. In: *SIGMOD Conference*. DOI: http://dx.doi.org/10.1145/223784.223842; http://dx.doi.org/10.1145/568271.223842

Andler, S, Marcus, B, Gustavsson, S and **Mathiason, G** 2007 DeeDS NG: Architecture, Design, and Sample application Scenario. In: *Handbook of Real-Time and Embedded Systems*. Chapter 29. London: Chapman and Hall/CRC.

Aslinger, A and **Sang, H** 2005 Efficient Replication Control in Distributed Real-Time Databases. In: *Computer Systems and Applications, 2005*. The 3rd ACS/IEEE International Conference on IEEE.

Barreto, J and **Ferreira, P** 2010 Meaningful Metrics for Evaluating Eventual Consistency. In: *Euro-Par 2010. Parallel Processing 16th International Euro-Par Conference, Ischia, Italy on 31 August to 3 September 2010*. Proceedings. Part II-Springer Link.

Burns, A and **Wellings, A** 2001 *Real-Time Systems and Programming Languages*. Harlow, England: Addison-Wesley.

Buttazzo, G, Spuri, M and **Sensini, F** 1995 Value vs. deadline scheduling in overload conditions. In: *Real-Time Systems Symposium (RTSS), 1995*. Proceedings, 16th IEEE.

Chen, X, Mohamed, A S, Xiaofang, Z and **Aoying, Z** 2013 *Quality-aware schedulers for weak consistency key-value data stores. Distrib Parallel Databases*. York: Springer.

David, B, Sherif, S and **Liang, Z** 2013 Towards Comprehensive Measurement of Consistency Guarantees for Cloud-Hosted Data Storage Services. In: *Proceedings of the Fifth TPC Technology Conference on Performance Evaluation & Benchmarking (TPCTC013)*. York: Springer.

Farn, W, Li-Wei, Y and **Ya-Lan, Y** 2011 *Efficient verification of distributed real-time systems with broadcasting behaviors. Real-Time Syst*. York: Springer.

Fronczak, A, Holyst, J, Jedyank, M and **Sienkiewicz, J** 2002 Higher order clustering coefficients. Barabasi Albert networks. *Physica A*, 316(1): 688–694. DOI: http://dx.doi.org/10.1016/S0378-4371(02)01336-5

Golab, L, Johnson, T and **Shkapenyuk, V** 2009 Scheduling updates in a real-time stream warehouse. In: *Proc. of the IEEE international conference on data engineering*. Data Engineering, ICDE '09. IEEE 25th International Conference on. DOI: http://dx.doi.org/10.1109/icde.2009.202

Gustavsson, S and **Andler, F S** 2005 Decentralized and Continuous Consistency Management in Distributed Real-Time Databases with Multiple Writers of Replicated Data. In *IPDPS '05 Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) – Workshop 2 – Volume 03 IPDPS'05*. IEEE Computer Society.

Hababeh, I 2012 Improving network systems performance by clustering distributed database sites. *J Supercomput*, 59(1): 249–267. DOI: http://dx.doi.org/10.1007/s11227-010-0436-9

Hamdi, S, Salem, M B and **Bouazizi, R B** 2013 Management of Real-time data in Distributed Real Time DBMS using Semi-Total replication data. In: *Computer Systems and Applications (AICCSA).* ACS International Conference on IEEE. DOI: http://dx.doi.org/10.1109/aiccsa.2013.6616447

Harder, T and **Reuter**, **A** 1983 Principles of transaction-oriented database recovery. *ACM Computing Surveys (CSUR)*, 15(4). DOI: http://dx.doi.org/10.1145/289.291

Ivanova, M, Kersten, M and **Nes, N** 2008 Adaptive segmentation for scientific databases. In: *Data Engineering, 2008*. ICDE 2008. IEEE 24th International Conference on.

Jiantao, W, Song, H, Kam-Yiu, L and **Aloysius, K** 2012 Maintaining data temporal consistency in distributed real time systems. *Real-Time Syst*, 48(4): 387–429. DOI: http://dx.doi.org/10.1007/s11241-012-9150-4

Jiawei, H and **Micheline, K** 2006 *Data Mining: Concepts and Techniques*. 2nd ed. York: Elsevier.

Jon, O, Norvald, H and **Kjetil, N** 2010 DYFRAM: dynamic fragmentation and replica management in distributed database systems. *Distrib Parallel Databases*, 28(2): 157–185

Kumar, P, Krishna, P, Bapi, R and **Kumar, S** 2007 Rough clustering of sequential data. *Data KnowlEng.*, 63(2): 183–199. DOI: http://dx.doi.org/10.1016/j.datak.2007.01.003

Laarabi, M, Boulmakoul, A, Sacile, R and **Garbolino, E** 2014 *A scalable communication middleware for real-time data collection of dangerous goods vehicle activities*. Transportation Research Part. York: Elsevier.

Lin, W and **Veeravalli, B** 2006 A dynamic object allocation and replication algorithm for distributed systems with centralized control. *Int. J. Comput. Appl.,* 28(1): 26–34. DOI: http://dx.doi.org/10.2316/Journal.202.2006.1.202-1409

Mathiason, G, Andler, F S and **Son, H S** 2007 Virtual Full Replication by Adaptive Segmentation. In: *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE computer society RTCSA 2007. IEEE.

Meixing, L, Krishna, K and **Sushil, J** 2014 Consistency and enforcement of access rules in cooperative data sharing environment. *Computer & security*, 41: 3–18. DOI: http://dx.doi.org/10.1016/j.cose.2013.08.011

Said, A H, Sadeg, B and **Ayeb, B** 2009 The DLR-ORECOP Real-Time Replication Control Protocol. IEEE. In *Emerging Technologies & Factory Automation, 2009*. ETFA 2009. IEEE Conference on.

Saito, Y and **Shapiro, M** 2005 Optimistic replication. *ACM Comput. Surv.*, 37(1): 42–81. DOI: http://dx.doi.org/10.1145/1057977.1057980

Shapiro, M, Bhargavan, K, Chong, Y and **Hamadi, Y** 2004 *A formalism for consistency and partial replication.* Technical Report MSR-TR-2004. Microsoft.

Song, H, Kam-yiu, L, Jiantao, W, Sang, H and **Aloysius, K** 2012 Adaptive co-scheduling for periodic application and update transactions in real-time database systems. *The Journal of Systems and Software*, 85(8): 1729–1743. DOI: http://dx.doi.org/10.1016/j.jss.2012.03.055

Srikanth, J and **Prasanta, K** 2014 Energy Efficient Grid Based Clustering and Routing Algorithms for Wireless Sensor Networks. In: *Fourth International Conference on Communication Systems and Network Technologies(CSNT)*. IEEE.

Terry, D, Theimer, M, Petersen, K, Demers, A, Spreitzer, M and **Hauser, C** 1995 Managing update conflicts in Bayou, a weakly connected replicated storage system. In: *Proceedings of the 15th ACM Symposium on Operating Systems Principles*. SOSP '95 Proceedings of the fifteenth ACM symposium on Operating systems principles, pp. 172–182. DOI: http://dx.doi.org/10.1145/224056.224070; http://dx.doi.org/10.1145/224057.224070

Wang, F, Li-Wei, Y and **Yang, Y** 2011 Efficient verification of distributed real-time systems with broadcasting behaviors. *Real-Time Syst.*, 47(4): 285. DOI: http://dx.doi.org/10.1007/s11241-011-9122-0

Xiang, Z, Haichuan, S H, Wenjie, Z, Xuemin, L and **Weidong, X** 2013 On Efficient Graph Substructure Selection. DASFAA 2013 In: *The 18th International Conference on Database Systems for Advanced Applications*. Springer – Volume 7826.

Xiong, M, Han, S, Chen, D, Lam, K and **Feng, S** 2010 Desh: overhead reduction algorithms for deferrable scheduling. *Real-Time Syst.*, 44(1): 1–25. DOI: http://dx.doi.org/10.1007/s11241-009-9087-4

Yan, C, Tilman, W and **Weibo, G** 2011 Delaying Transmissions in Data Communication Networks to Improve Transport-Layer Performance. *IEEE Journal on Selected Areas in Communications*, 29(5): 916–927. DOI: http://dx.doi.org/10.1109/JSAC.2011.110502

How to cite this article: Salem, R, Saleh, S S and Abdul-kader, H M 2016 Scalable Data-Oriented Replication with Flexible Consistency in Real-Time Data Systems. *Data Science Journal,* 15: 4, pp. 1-15, DOI: http://dx.doi.org/10.5334/dsj-2016-004

Submitted: 21 August 2015 **Accepted:** 10 December 2015 **Published:** 17 March 2016