International Journal of System Dynamics Applications
Volume 10 • Issue 4 • October-December 2021

A Multi-Replica-Centered Commit Protocol for Distributed Real-Time and Embedded Applications

Anupama Arun, Madan Mohan Malaviya University of Technology, Gorakhpur, India
Sarvesh Pandey, Banaras Hindu University, Varanasi, India
https://orcid.org/0000-0002-3014-9792
Udai Shanker, Madan Mohan Malaviya University of Technology, Gorakhpur, India
https://orcid.org/0000-0002-4083-7046

ABSTRACT

Modern multi-site database applications are not only time-driven but also require efficient quality of services with no single-node failure. It might be ideally achieved using database replication techniques. The transactions, being a basic component of these applications, are more likely to miss their deadlines because of requiring an unpredictably long time to access remote data items. The temporal validity of data is another issue requiring attention to be paid. To address these problems, a cluster-replicas with efficient distributed lazy update (CRED) protocol is proposed in this paper. The CRED protocol increases the chance of timely execution of transactions and data freshness in an unpredictable workload environment by utilizing the lazy replica update strategy. It reduces the negative impact of the burst workload with a marginal overhead of ensuring timely updated replicas. The simulation results confirm that the CRED outperforms the ORDER protocol by up to 4%.

KEYWORDS

Data Replication, Distributed Real-Time Database, Immediate Update, On-Move Transaction, Refresh Transaction

INTRODUCTION

In today's digital era, there are countless real-world applications requiring access to the remotely distributed data present at multiple distant sites (Omamo, Rodrigues, & Muliaro, 2020) (M., K., & K., 2020). Telecommunication services and online trading systems are some of the easy-to-understand example applications (Mustafa, 2021) (Gupta & Shanker, 2020). Here, data is dispersed across geographically distant sites and a wide-area network is utilized to communicate among these sites (Pandey & Shanker, 2016) (Srivastava, Shankar, & Tiwari, A protocol for concurrency control in real-time replicated databases system, 2012). All such applications exhibit characteristics like a fast exchange rate with data access from anywhere at any time (Pierce, Shepherd, & Johnson, 2019) (Kizito & Semwanga, 2020). Besides, the location of a node/device can also be useful for some custom applications (Gupta & Shanker, 2021). With increased application complexity, the underlying data and transactions accessing that data both are associated with time constraints or deadlines. In simple words, one needs to take care of the fact that the data deadline is also honored in addition to the

DOI: 10.4018/IJSDA.20211001.oa18

transaction deadline (Ulusoy, 1994) (Singh, Pandey, & Shanker, 2019). These applications require data that is constantly changing — weather data, stock prices, health indicators etc. So, data generated by such applications have some validity associated with it. This means that data can be used to take some action using transaction only before the expiry of its validity.

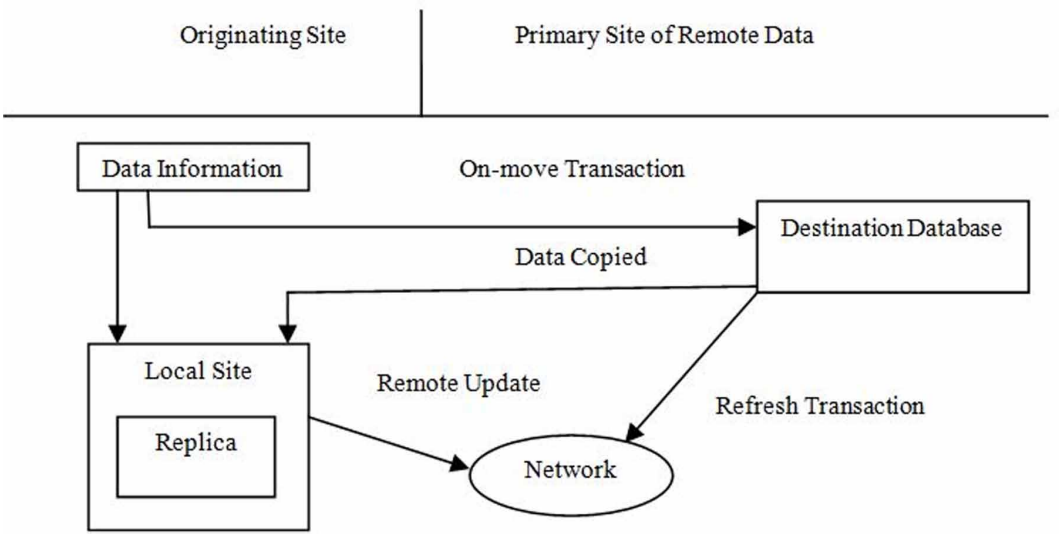
The above-discussed applications are studied under the umbrella of a research area named "Distributed Real-time Database Systems (DRTDBS)". The DRTDBS is a finite set of geographically separated database sites connected through a network (Pandey & Shanker, 2020) (Shanker, Misra, & Sarje, 2006) (Pandey & Shanker, 2018). Each database site consists of one local database. Considering the whole set up as one logical unit, any such change in the state of data could be performed using real-time distributed transactions. A real-time distributed transaction is a real-time transaction that requires access to both local and remote data items. Going one level further down from a definition perspective, a real-time transaction is a transaction with associated time-constraint/deadline. The two key issues with DRTDBS based applications are deadline miss of transactions and expiry of data deadline. These issues mainly occur because of longer data access latency. A transaction processing framework mainly consists of the following components — priority assignment scheme (Pandey & Shanker, 2019), concurrency control (Pandey & Shanker, June 18-20, 2018) (Pandey & Shanker, 2017), and commit processing (Pandey & Shanker, 2018) (Pandey & Shanker, 2019). Replication of data requires adjustment in all the above 3 components.

Data replication could be an effective solution to address the above problems as it increases the availability of data. One approach to address temporal data validity expiration related issue is by creating a replica of temporal data at a site requesting access to it often. In this way, a local replica of data present at a remote location can be accessed faster — no need of waiting for the remote site all the time for executing a transaction. It ensures that the transactions meet their deadline before the expiry of the temporal validity of data. There are mainly two approaches for implementing replication: full and partial. The replication approach should be chosen based on the criteria such as the type of workload and specifications of the database. Irrespective of the approach chosen, the algorithmic design problem is to come up with an efficient process to keep replicas available at multiple sites updated. This creates an overhead. Moreover, keeping replicas updated is very costly in terms of performance degradation at a higher workload with unpredictable data access patterns. More and more transactions miss their deadline with the increase in workload.

Contribution: This paper proposes the CRED protocol based on replication. It comes up with a better mechanism to access temporal data (local & remote). On the invocation of a transaction, following information is declared — deadline, pre-requisite data set access list, execution time, and validity period. Here, data items are replicated based on the parameters of a transaction originated at any given site. Using the transaction and other meta-data information, on-move transaction creates a copy of the data present at the primary site to the remote. This way, using the replica of data, all further processing is done at local site only — data modification is done locally with the help of on-move transaction. Because of the on-move transaction, not only the site which is currently having a copy of original data gets benefited; but, the other sites, which are in the neighborhood or the cluster, are also benefited from this copy of data. However, the synchronous replica update delays commitment of the transaction.

Therefore, the concept of a lazy-master update is used to update a replica. In the lazy-master update, the remote site replica is updated by a new transaction called the refresh transaction. In this strategy, they accumulate a set of modifications and propagate to all the sites where the replica of data is available. In this way, there is a reduction in the overhead incurred. With this algorithm, the commitment time of the transaction is reduced; therefore, the large-scale distributed transaction meets its deadline and maintains the temporal validity of data also. Compare with other replication control algorithms, CRED can upgrade the system performance when system size increases, as well

Figure 1. The Logical Representation of CRED Protocol



as the workload is very high with the presence of unpredictable data access patterns. In Figure 1, the logical representation of the CRED protocol is presented.

Organization: The remainder of this paper is organized as follows. The immediate next section summarily describes the literature review of the domain of the work reported here. Further, next section offers the distributed real-time replica-based on-demand distributed real-time data model that was implied to implement the discussed protocol. In next section, the CRED protocol is proposed and discussed in detail. The outcome of the performance study is presented and discussed in second last section. We analyze the performance of our protocol through a simulation study. Finally, conclusions & future scope for the research domain are written in last section.

BACKGROUND AND RELATED WORKS

The transaction processing in the DRTDBS environment may suffer from miss of transactions deadline and expiry of data-deadline due to remote data access patterns. To maintain temporal as well as the logical consistency of the data and to avoid transaction deadline misses, one can use the replication technique that enables the availability of data, reduces response time of the transaction, and facilitates concurrent execution of queries. Therefore, the database replication can be defined as a process of creating a mirror copy of a database — the replica of the database acts as a slave database while the source or primarily existing database acts as a master database. It can be mainly categorized as snapshot replication (simple mirroring of a database), merge replication (the Master-Slave type of replication where the entire database resides at master and subset databases reside at multiple sites), and transactional replication (considering the role of a transaction while replicating data objects). The utilization of any such data replication technique results in qualitatively increased data availability, improved read access performance, etc. Though the mirroring of a database increases the data availability, it is sometimes difficult to integrate the transactional replication strategies considering the nature of applications. It comes at a cost of requiring larger storage space and overhead of maintaining multiple copies of data compared to just one at different database sites. Therefore,

nowadays, researchers are putting more effort into integrating the notion of database replication technique with the transaction concept.

The replication techniques could be broadly categorized as of two types —full and partial. In full replication, all the sites in the distributed system maintain one copy of data, i.e., data items are fully replicated across sites. The advantage of this technique is that the data items are accessible from all sites and read operation on data is done locally; however, ensuring the updated copy of data at each site turns out to be very costly in large scale database systems. Moreover, updating replica copies becomes a bottleneck at a higher workload. This results in a performance degradation because of replication at a higher workload — this is a perfect example of a scenario where a solution becomes a problem. Partial replication is a replication strategy, where data is replicated at a small number of sites only. It is best suited for medium-scale and large-scale databases. Here, the replication of the data is done as per the need of the transactions. The data replication techniques also suffer from various other problems. Logically, all the copies of the replicated data must have the same value. This means every time any update is performed on a data item, all its replica copies (available at other geographically distant sites) must be updated accordingly to ensure the valid state of data across sites. Updating multiple copies of the same data at different sites leads to overheads — communication overhead and network congestion.

The replication, in general, can be active (all the sites are updated synchronously) or passive (all the sites are updated asynchronously) in nature. The complexity involved in implementing and utilizing the notion of database replication is dealt with using group communication mechanisms. More specifically, the choice of using an active or passive strategy comes up with its own set of positivity and challenges. With active replication, one can save a lot of implementation efforts as it is easy to implement and failure transparent. By failure transparency, it is meant that the end-user should not be aware of the situation where he was switched to another replica because of the failure of some replica that he was accessing. In this way, a replication approach helps the developers to keep the product running and available even with a scenario of few node failures. Moreover, a passive replication strategy tries to reduce the bandwidth overhead of keeping all the replicas fully updated all the time. In the process of doing so, it requires only the master replica to invoke the operation and gets updated based on the transactional/logical requirements. All the slave replicas are updated asynchronous way and sometimes in a piggybacked manner to reduce the communication overhead. This also helps the running transaction to complete its execution faster by providing better utilization of the communication channel. However, dealing with communication breakdown is extensively difficult with the passive replication-based strategy compared to the active replication strategy. Here, the primary replica site and other slave replica sites need to make sure that updates are processed in the same logical order.

There are several variants of 2-phase locking (2PL), timestamp ordering, as well as optimistic approaches for distributed concurrency control (CC), developed for the detection and resolution of conflict (Pandey & Shanker, 2020). But, they suffer from severe damages caused by deadlock and priority inversion. A large number of research works have been reported for the handling of data replication in the case of traditional database systems (Son, 1988). In past years, numerous research-works for real-time databases have been published (Srivastava, Shankar, & Tiwari, 2012). However, relatively less effort is devoted to design replication control algorithms for DRTDBS. There are only a few concurrency control algorithms invented to solve the issue of conflicts in replicated DRTDBS.

The replication algorithm can also be categorized in static as well as dynamic replication algorithm. The MIRROR (Managing Isolation in replicated real-time object repositories) (Xiong, Ramamritham, Haritsa, & Stankovic, 2002) is a CC algorithm designed for the DRTDBS utilizing a replicated strategy. In this algorithm, optimistic two-phase locking (O2PL) is supplemented by the inclusion of the priority abort & priority block conflict mechanism. The prominent idea of this algorithm is to prevent data conflicts based on the states of a distributed transaction. In (Peddi & DiPippo, 2002), Peddi et al. have presented a replication algorithm, which copies the remote site data

to the needed site by using the concept of the replication transaction and provides a guarantee for the read of temporal valid data. This algorithm is for a static environment, where the data requirement is expected to be known a priori. The ORDER (On-demand Real-time DEcentralized Replication) algorithm (Wei, Aslinger, Son, & Stankovic, 2004, August)is developed for an environment where all the data types and their relations are expected to be known a priori or well in advance, and transactions are of short-term periodic. The ORDER-RS (ORDER-replica sharing) is an extended version of the ORDER. This replication scheme enhances the system performance when the system consists of a large number of distributed database nodes. Here, sites are divided into clusters, and replicas are shared within the cluster. The notion of "Virtual full replication" was proposed to achieve scalability affairs. In virtual full replication, the database is divided into segments and different segments have a different number of replications. An algorithm is designed to guarantee the quality of services of temporal data by applying full replication in DRTDBS of small scale only. The two algorithms were proposed to work under dynamic periodic and aperiodic workload environments. The algorithm uses partial replication with a dynamic environment to fulfill the unpredictability of dynamic requests.

The Real-Time Replication Control Protocol (RT-RCP) (Said, Sadeg, Amanton, & l Ayeb, 2008) proclaims that the update of a replica is done immediately when a transaction has enough time to complete updates within its time limit; otherwise, it is done after completion of the commitment process of the original transaction. The algorithm is designed by the augmentation of the data update process as well as the commitment process to avoid the extra steps for the propagation of the updates. With the DLR-ORECOP algorithm (Said, Sadeg, Ayeb, & Amanton, 2009), updates of a replica of data items do not lead to an increase in the execution time of the transaction. Gustavsson et al. (Gustavsson & Andler, 2005), introduce the convergence protocol for DRTDB for multiple updates on replicated data. The algorithm is designed for a system, where local predictability and performance of the system are more important than global consistency. Salem et al. (Salem & Abdul-kader, 2016) address the scalability issues in the DRTDB by using the dynamic clustering technique. In this algorithm, the new updated technique for solving the temporal inconsistency is removed by skipping unnecessary operations by allowing the database sites to update data simultaneously without looking for synchronization.

In the context of the DRTDBS, the transaction scheduling schemes suffer from various other problems such as data freshness, application scalability, prolonged remote data access time, network partitioning, fault tolerance, etc. All such problems can be addressed to some extent by utilizing appropriate application-specific replication strategies. However, integrating replication strategies with transaction scheduling protocols results in another set of problems, which requires further attention of the database research community. Furthermore, conflict resolution can be one of the areas to focus on as the way conflict is resolved considering no replication is completely different. So, the existing conflicting resolution strategies need to be reinvestigated considering the more complex replicated environment which is the need for today's applications. Researchers are expected to come up with solutions such as broadcasting mechanisms, and consensus protocols to facilitate the integration of the replication strategies with the DRTDBS applications.

REPLICA AWARE PARTITIONED DRTEDBS MODEL

The finite set of database sites connected through a network is assumed to form a distributed database system. Each transaction (and data) has a time-constraint associated with it. This setup is collectively named as a distributed real-time and embedded database system (DRTEDBS) and applied to manage the real-time data of day-to-day applications such as e-commerce, online trading, stock marketing etc. Here, data objects have a period of validity associated with them after which they are unusable. A firm deadline-based replicated DRTEDBS model is considered for this study. This means transactions are aborted if they do not meet the deadlines (transaction and data deadline).

Modelling Data and Transaction

The data at the original site is called primary copy while the same replicated at other sites is called a replica of it. Modeling of data can be customized based on its characteristics — temporal and non-temporal. Temporal data mostly get generated from small sensor devices. These sensors can be used to monitor physical world activities like building structure health, air quality index, temperature etc. The validity interval associated with each temporal data object is called an absolute validity interval (AVI). The AVI is a period during which data remains valid (Stankovic, Ramamritham, & Towsley, 1991). The temporal data becomes stale if the difference between the current time and data timestamp is greater than the data AVI. This is not the case with non-temporal data. It needs not to get updated on a periodic basis.

Each transaction has attributes such as deadline, data access list, and execution time. The priority assignment policy of the transaction processing framework assigns priority to transactions. All further scheduling is done based on transaction priority. Two types of transactions are considered in this study — update transaction and user transaction. The update transaction is a periodic transaction requiring temporal data for access. Moreover, a temporal transaction is always given higher priority compared to the user transaction. The reason behind giving higher priority to update transactions over user transactions lies in the fact that user transactions can request only read access to temporal data while it can request both read and write access to any non-temporal data. The goal is to finish the execution of the transaction before its deadline expiry while making sure that the temporal data accessed remains valid (temporally consistent) as well till the final outcome of transaction.

The temporal consistency is defined for temporal data. The value of temporal data is always versioned. The i^{th} version of temporal data d_{t_i} can be represented as below:

$$i^{th}\ version\ of\ d_{t_i} = \left(value\ of\ d_{t_i}, temporal\ validity\ interval\right) \tag{1}$$

where i = 1, 2, …, n.

The transactions are scheduled based on the priority assignment policy discussed here. This policy assigns the priority based on the minimum of the data-deadline and transaction deadline. At time t, the priority of a sensor transaction can be computed as below:

$$P_t\left(T\right) = MIN\left(dd_t\left(T\right), r_t\left(T\right)\right) \tag{2}$$

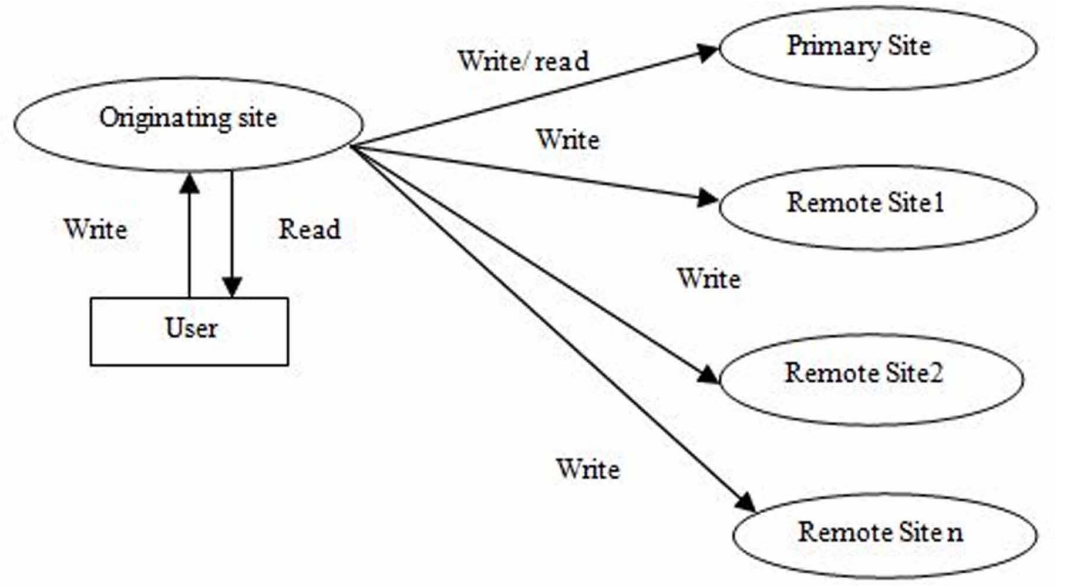
where $P_t\left(T\right)$ denotes priority of transaction at time t, $dd_t\left(T\right)$ is data-deadline of a transaction at time t, and $r_t\left(T\right)$ denotes the estimated remaining execution time of the transaction at time t.

A periodic sensor transaction updates the associated temporal data items at the frequency F_i. All these temporal data items come under a write set (W_i). After updating the primary copy, all other replica copies of it available at k different sites are updated. The bandwidth utilization is computed as the number of messages communicated per second. This can be represented using the below equation:

$$\text{Bandwidth Utilization} = \left(k-1\right)\sum_{i=0}^{k} W_i * F_i \tag{3}$$

Suppose that there are multiple sites available in the DRTEDBS and each site contains the temporal and non-temporal data. If transaction requires access to the remote data, the replica of it

Figure 2. Replica Update in ORDER Protocol



(remote data) would be created at a local site. With full replication, the process of creating a replica leads to a higher bandwidth utilization value compared to any other replication strategy.

CRED-A MULTI-REPLICA CENTERED COMMIT PROTOCOL

The main performance issues with data replication strategies in the DRTEDBS are as follows. At first, it is required to ensure that stringent temporal constraints associated with data and transactions are met. Second, the process for updating replicas available at multiple sites should be customized (as per the application's need). Various replication schemes are designed, in past, to handle different data access workloads as well as database specifications.

Accessing remote data is a costly multi-hop operation as it requires considerably higher time while accessing the local one is comparatively less costly involving just a single hop. If not properly dealt with, this might result in increased transaction deadline miss percent. In addition to the above, if the remote data item is a temporal one then the chances of completing a transaction requiring access to it become even lesser. It is because accessing remote data item naturally increase transaction execution time, and in case of a temporal remote data item, the validity as well needs to be ensured. The work reported in (Wei, Aslinger, Son, & Stankovic, 2004, August), addresses these issues to some extent with a focus on replicating the data as per the demand of the transaction. When a transaction is generated at a site, the data needed by it must be evaluated. Based on the outcome of this evaluation, the ORDER algorithm determines dynamically that where and how often replicas are to be updated. Figure 2 represents the working idea of the ORDER protocol.

In case of requiring access to temporal data, the transaction needs to access a fresh/valid version of it only. The validity of temporal data expires at a regular interval — the new version of value is assigned every time its previous value expires. Updating all the copies of the temporal data available at distinct sites is a time-consuming operation. Because of requiring multiple message communications for synchronous replica updates and timely & fast-speed access to remote temporal data, the performance of the DRTEDBS needs to be assessed more seriously. These requirements might result in larger transaction commit time, and higher transaction miss percent. To address the

above issues, the CRED protocol is designed that uses the on-move transaction for accessing remotely available data items.

CRED REPLICATION PROTOCOL

The goal of the CRED protocol is to meet the stringent temporal constraints (for instance, transaction deadline and data-deadline) while updating the replicas of modified data items at distributed sites. This protocol reduces the replica update overhead in a distributed environment considering the bursty workload scenario. The proposed replication strategy performs well compared to existing replication strategies (full and partial). This has been using the novel on-move replication strategy. In the considered database model here, sites contain the temporal / non-temporal data as well as their replicas. The temporal data is updated by the periodic system update transaction at a specific basis of update frequency. The replica of the temporal data — present at the primary site and updated after the expiry of current temporal validity — is propagated to other sites at extended update frequency through a special transaction named as refresh transaction.

Each transaction, irrespective of its type, has the following information tagged with it — execution time, deadline, pre-requisite data items list. Going one level further down, in case the data item is a temporal one, the following information would be tagged with it — data deadline, absolute validity interval (AVI), data freshness requirement (FR), and site ID of the primary site that has a primary copy of data.

In the CRED, the on-move transaction and the refresh transaction are responsible for providing data at the needed site and updating the modified data at remotely existing sites, respectively. When a site receives a local transaction to be executed, the designed algorithm determines the data needs of the transaction and evaluates the remote data requirement. Based on data information needs by the transaction, the on-move transaction is created to copy the data to the needed site and update the modified data to the site from where it is fetched. By the on-move transaction, not only the site which is currently having a copy of the original data item getting benefited but the other sites that are in the neighborhood or are in the cluster, are all benefited from this copy of data. The updates of the other replica sharing sites are updated by the refresh transaction after the commit of the on-move transaction. The update of the replica is performed by using the concept of immediate lazy update. Suppose that algorithm receives a request to access a remote data D available at site X, it is accomplished as per steps given in the pseudo-code of the algorithm-I for the replication.

As the transaction is invoked at the site, the transaction's parameters are known a priori. If the transaction wants to access remote site data, then first of all checking is done whether the replica of that data is present locally or not. If the replica is present at the local site, testing is done for the extended update frequency of the data. If the required current update frequency is less than the extended update frequency, then the transaction can access that data. In case, if a replica of data is not present locally, then the replication manager creates a transaction called an on-move transaction, which copies the data replica from the primary site to the needed site. The modified data is updated at the primary site by this on-move transaction. In this scenario, the replica's closing time is also modified with the sum of the current time and duration of the newly arrived transaction. If the replica is available, but its extended update frequency is less than the required update frequency, the on-move replication is already present at that site which updates the new version of the update frequency from the primary site of that data item. The modified data at the primary site is pushed back to other remote sites by other transaction called refresh transactions in an asynchronous way.

The update process follows the Immediate-Lazy update approach, in which the modified data copy is propagated inside the message after the commitment of the transaction which modified the data. Due to this, the inconsistency/freshness of data is maintained at each site. The update transaction is waiting for the commitment of the original transaction, other site updates of replicas are performed simultaneously. This strategy reduces the commit time of the transaction and the transaction meets its

deadline as well as commits before the expiry of the data-deadline. The pseudo-codes for computing update frequency & duration and updating replicas at other remotely distributed sites from the primary site are discussed below. It is also clear from the below algorithms that how the replica closing time calculation is performed.

Algorithm 1. For Calculation of Update Frequency and Duration

Input: $E_{curr}(d, s)$ is a current Extended Update Frequency (EUF) for a temporal data item d at site s. $E_{up}(d, s)$ is the updated EUF for temporal data item d at site s. $CT_{curr}(d, s)$ is the current replica closing time for temporal data item d at site s.

```
If (remote data item access request):
{
        If (active replica):
     {
                If (Ecurr(d, s) ≥ Eup(d,s)):
                        {
                        Ecurr(d, s) = Ecurr(d, s)
                        Use the Ecurr(d, s). No action is
required.
                        }
Else
        {
        Ecurr(d, s) = Eup(d, s)         // Use the Eup(d,s)
        CTcurr(d,s) = CTcurr(d,s) + incoming transaction request
duration
        On-move-transaction-creation (d, s, duration, execution
time, deadline)
        // On move transaction is created to replicate data from
primary site.
        ActiveReplicaRegsitration(Ecurr(d, s), CTcurr(d,s))
        }
                        }
                Else          // No active replica available
                      {
                        ActiveReplicaCreation(d,s)
Ecurr(d, s) = Eup(d, s)         // Use the Eup(d,s)
CTcurr(d,s) = CTcurr(d,s) + incoming transaction request duration
On-move-transaction-creation (d, s, duration, execution time,
deadline)
ActiveReplicaRegsitration(Ecurr(d, s), CTcurr(d,s))
                                        }
                }
```

Algorithm 2. For the Role of Refresh Transaction in Replica Update Propagation

Input: $\mathbf{Log_{IN}}$ is an input log consisting of a set of status of operations on data items. Message m is the smallest message unit that carries the updated data item d which is the smallest

```
database unit. The M_i carries the information about the
sequence of updated data items associated with the same refresh
transaction.
Output: The message is sent to active replicas through the
refreshment transaction.
For each updated data item d_i:
        Read (Log_IN, o)
        If (operation on d_i is write and transaction T is new):
                Message M is created for T.
        Else If (operation on d_i is write and transaction T is not
new):
                The updated data item d_i is added to M.
 // When transaction reaches its final decision, the system
decides about message M.
        If (final_decision == COMMIT)
                Propagate message M to all the required active
replicas
        Else
                Discard the message M as it is no longer
required.
```

As can be interpreted from algorithm II, when an update is performed, a new transaction called the refresh transaction is created for the update of the replica at a different site from the primary site. The updated information is read from the reception log by the receiver and the refresh transaction is created to update the replica. If the update is coming from new transactions to update the same data, then it is stored in a new queue and update serially as the message for the update has arrived. In case, if the original transaction is aborted due to some reason, then all the modifications, which are done at the remote site, are also discarded to prevent the other transaction to read dirty data.

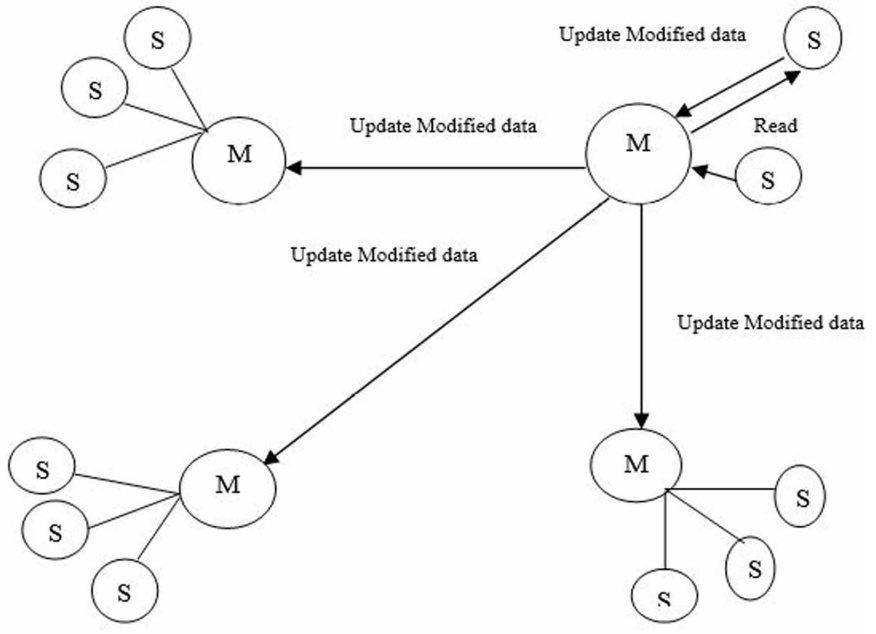
The CRED protocol provides the data needs from a remote site to the local site and updates only the one site from where the data is copied after the modification of the data item. The rest of the replica sharing sites are updated by the refresh transaction after the commitment of on-move transactions. By the above concept, not only the overhead of updates at multiple sites is reduced, but it also guarantees the timely execution of the transactions and data freshness in cases of the unpredictable workload and data access pattern.

Partitioning of Large-Scale Database to Reduce Update Cost of Replica

The CRED algorithm has also analyzed best in large scale databases where the cost of the update of the replica has increased. First, in a large-scale distributed database, it might be very costly to update the modified data at each replica. So, the reduction of cost has been done by partitioning the large-scale distributed database region-wise (see Figure 3). In each region, there will be a master database node that has both temporal and non-temporal data. The master node has a collection of sub-nodes called slave nodes where replica copies are available only for performing read operations locally. If the modification of data has been done in one slave node under the same master node then the update has been performed on masters only. The update will not be propagated to all slave nodes. The slave node accesses the modified data when it is required.

If any request has been invoked for accessing the data object, then first of all checking for the presence of replica locally is done. If the response is yes, then a comparison of the absolute validity of the replica is done. If requested data replica validity is greater than the present replica validity, then discard the locally available replica. The demand for accessing the data is fulfilled by the on-move transaction. The on-move transaction copies the replica from the master node to the slave node and copies back the modified data to the same master node. The update of modified data has been

Figure 3. Logical View of Replica Update in Large Scale DRTDBS

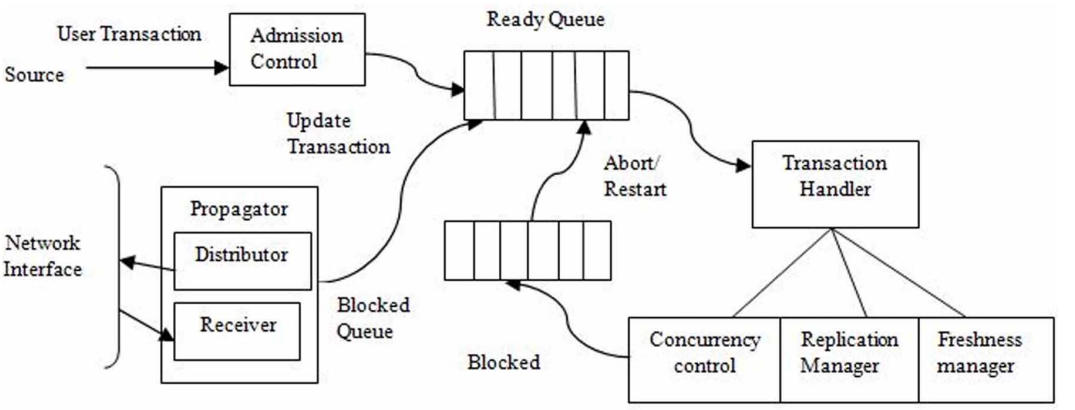


performed by the Immediate-Lazy update to each master node by refresh transaction. Partitioning the large-scale distributed database into region wise and allotment of the master node in each region reduces the cost of the update of the replica. In this way, we do update on the master node only; there is no need to update all the nodes where replicas are present.

SIMULATION AND RESULT OBSERVATIONS

To evaluate the performance of the proposed protocol, a Replicated DRTEDBS model is logically visualized and then simulated. Our model is influenced by and similar to the models presented in (Ulusoy O. a., 1992) (Ulusoy & Belford, Concurrency Control in Real-Time Database Systems, 1992) (Ulusoy & Belford, 1992) (Ulusoy Ö., 1995) (Ulusoy & Belford, Real-time transaction scheduling in database systems, 1993). The architecture of simulation consisting of components such as admission control, transaction handler, and propagator is given in Figure 4 (Pandey & Shanker, 2020) (Pandey & Shanker, 2020). The admission control component is used to minimize the chance of nodes to be overloaded because overloading of the system can lead to unwanted consequences such as transaction deadline miss and less utilization of CPU resources. However, admission control is applied for only user transactions. The transaction handler is constituted by the integration of the concurrency controller, freshness manager as well as replication manager. The transaction may be aborted and/or restarted due to the decision taken by the concurrency controller component. The replication manager manages the on-move transaction by analyzing the data set required by the corresponding transaction. If replicas of the remote data objects are not locally present, then the on-move transaction copies the remote site data to the needed site. After receiving the commit message from the corresponding transaction, the modified replica data is updated at the parent site of the corresponding data object, and finally, the transaction is ready to commit without updating the corresponding replica to shared sites. The freshness of temporal data is checked by the Freshness manager before the initiation of a user

Figure 4. Architecture of the DRTEDBS Model



transaction. If the freshness manager finds that data is stale, then the user transaction will be blocked and wait in the blocked queue as soon as the corresponding Update transaction will not be completed.

The propagator is a replication module that receives the replica updates and distributes the replica update to other shared nodes. The responsibility of the propagator is to receive the replica from other nodes and distributing it to other shared nodes.

The proposed model uses the lazy master approach, in which an immediate-immediate update strategy is used for guaranteeing data freshness. The immediate-immediate update strategy (Pacitti, Simon, & Melo, 1998) allows each write type operation to be performed by a transaction propagated with the help of transmitting the message after the commit operation completion of the original update transaction. The CRED protocol permits the update to be done on a parent node independent of the propagation of the replica copy. By this, the update is performed at a shared node. So, chances of the inconsistency or staleness of temporal data is reduced. Moreover, the transaction will not be kept in the blocked queue for the commitment of corresponding update transaction. This approach reduces the commitment time of transactions and increases the scalability of the system. The main objective of developing a simulator and performing experiments is to build the transparency that the proposed protocol improves the performance in terms of transaction miss percent. A list of parameters and their respective values are presented in Table 1.

A set of experiments were conducted to identify the impact of the proposed protocol compared to the traditional approach. The most important performance metric is Transaction Miss Percent (TMP), i.e., the percentage of transactions that are not completed within deadlines and eventually aborted. This is a conventionally used parameter metric to assess the performance of the replicated DRTEDBS (Pandey & Shanker, 2018) (Pandey & Shanker, 2020):

$$TMP = \left(\frac{\text{Number of transactions that have missed their deadlines}}{\text{Total number of transactions}} \right) *100 \tag{4}$$

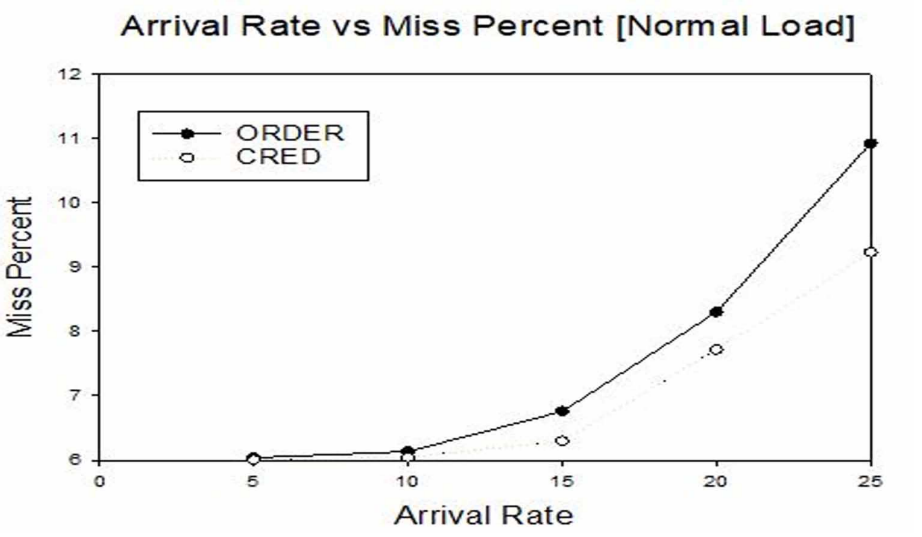
With the CRED protocol, the value of k is controlled intelligently which further results in the lower value of bandwidth utilization (see equation 3 for bandwidth utilization formula). It can be analytically confirmed as well that the CRED protocol outperforms other replication control approaches. By using this approach, the response time of the system is reduced as well as the number of messages generated is lesser. This approach is best suited under the large-scale distributed databases to guarantee the freshness of data as well as timely execution of the transactions.

Table 1. System Parameter Setting Representing User Transaction Workload

Parameter	Default Values
Number of Sites	10
Number of pages in each database	1000 pages
Degree of Replication	4
Number of CPU per site	2
Number of data disks per site	4
Log Disk at a site	1
Buffer Hit Ratio of a site	0.1
Transaction Execution Mode	Sequential
Transaction Arrival Rate	Variable
Slack Factor	6
Average transaction page access requirements	10 pages

It can be seen from Figure 5, transaction miss percent is considerably low for both the protocols which are obvious as normal load results in a lesser number of data conflicts. The CRED protocol performed only incrementally well as compared to the ORDER protocol because it handles the execution of global transactions efficiently. However, with a normal load, we have a smaller number of global transactions. It means, such an environment is not suitable to assess the features (capabilities) of the proposed CRED protocol.

Figure 5. The Effect of Varying Transaction Arrival Rate under Normal Load



It can be observed from Figure 6 that transaction miss percent is considerably high for both the protocols which are obvious as heavy load results in a larger number of data conflicts. The CRED protocol performed significantly well as compared to the ORDER protocol because it handles the execution of global transactions efficiently. Moreover, with a heavy load, we have a larger number of global transactions. Such an environment suits well in assessing the performance of the proposed CRED protocol.

Figure 6. The Effect of Varying Transaction Arrival Rate under Heavy Load

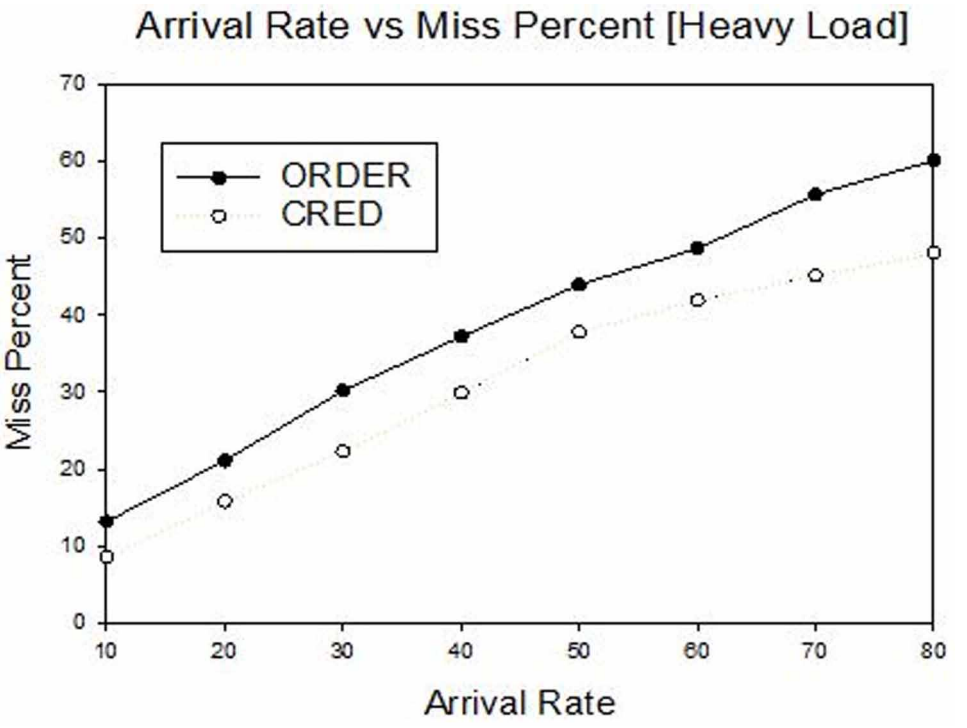


Figure 7 analyzes the performance of ORDER and CRED protocol considering the update frequency. As evident from the above figure, the increase in update frequency negatively affects the system performance, i.e. increased update frequency results in increased transaction miss percent. The main reason behind this behavior is that the communication and data freshness overhead increase with an increase in update frequency.

The system utilization for the ORDER and the CRED protocol are shown in Figure 8. As we can see that our proposed protocol provides better system utilization, i.e., lesser wastage of valuable system resources. CPU utilization increases with each remote data item accessed by the global transaction. On comparing with the ORDER protocol, the lower system utilization with CRED protocol is mainly because of better handling of remote data item access requirements using active replicas.

Figure 7. Effect of Varying Update Frequency on Transaction Miss Percent

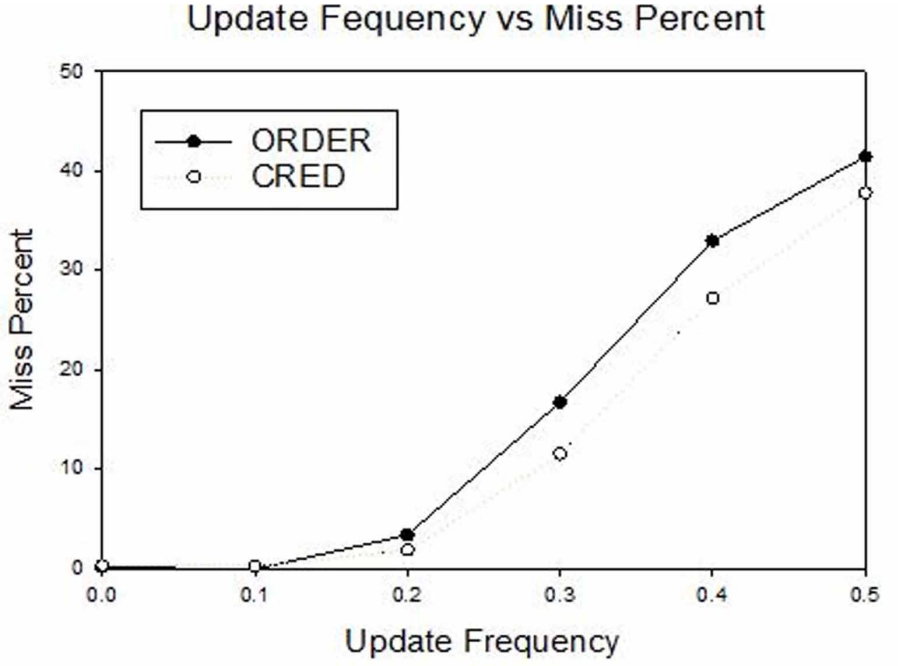
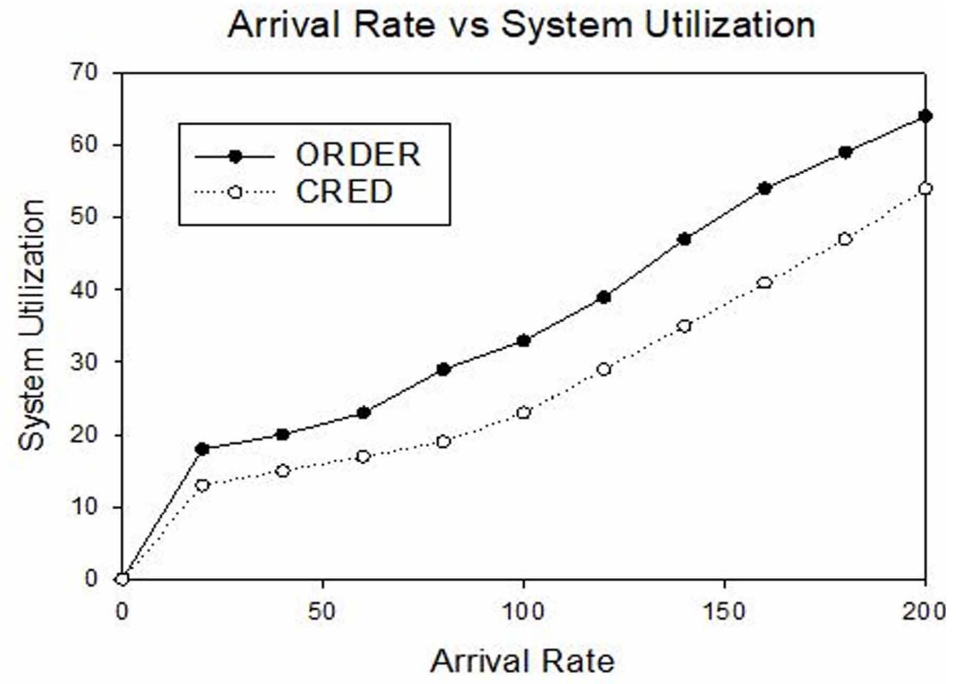


Figure 8. System Utilization for ORDER &CRED Protocol



CONCLUSION

In this paper, a dynamic replication algorithm has been presented — especially designed for medium and large scale distributed and embedded real-time database systems. Two types of transactions are introduced in the system, i.e., on-move transaction and refresh transaction. Temporal inconsistency and data freshness issue with periodic sensor transactions is addressed in the proposed CRED protocol. The unpredictable workload scenario is extensively evaluated to assess the suitability of protocol in a real-world scenario. The commitment time of the transaction is reduced by requiring a lesser number of messages in the transaction execution life cycle. Moreover, the freshness of data is maintained using an immediate update approach.

We restricted ourselves towards understanding the behavior of transaction replication strategies with an assumption that the transaction is a flat transaction type (Shanker, Misra, & Sarje, 2006). However, there are many other real-time complex models available such as nested transactions, the concept of active transactions, in-memory transactions, etc. In the future, to address the wider needs of the research community, we will work on to come up with a logical extension of ideas presented for the flat transaction model to other transaction models (Pandey, Pandey, & Shanker, 2019) (Pandey, Pandey, & Shanker, 2020) (Ozsoyoglu & Snodgrass, 1995).

REFERENCES

Gupta, A. K., & Shanker, U. (2020). OMCPR: Optimal Mobility Aware Cache Data Pre-fetching and Replacement Policy Using Spatial K-Anonymity for LBS. *Wireless Personal Communications*, *114*(2), 1–25. doi:10.1007/s11277-020-07402-2

Gupta, A. K., & Shanker, U. (2021). Prediction And Anticipation Features Based Intellectual Assistant in Location Based Services. *International Journal of System Dynamics Applications*.

Gustavsson, S., & Andler, S. R. (2005). Continuous consistency management in distributed real-time databases with multiple writers of replicated data. *19th IEEE International Parallel and Distributed Processing Symposium*. doi:10.1109/IPDPS.2005.152

Kizito, A., & Semwanga, A. R. (2020). Modeling the Complexity of Road Accidents Prevention: A System Dynamics Approach. *International Journal of System Dynamics Applications*, *9*(2), 24–41. doi:10.4018/IJSDA.2020040102

M., A., K., R., & K., K. (2020). Cloud-Based Access Control Framework for Effective Role Provisioning in Business Application. *International Journal of System Dynamics Applications, 9*(1), 63-80.

Mustafa, N. (2021). Research and Statistics: Coronavirus Disease (COVID-19). *International Journal of System Dynamics Applications*, *10*(3), 1–20. doi:10.4018/IJSDA.20210701.oa1

Omamo, A. O., Rodrigues, A. J., & Muliaro, W. J. (2020). A System Dynamics Model of Technology and Society: In the Context of a Developing Nation. *International Journal of System Dynamics Applications*, *9*(2), 42–63. doi:10.4018/IJSDA.2020040103

Ozsoyoglu, G., & Snodgrass, R. T. (1995). Temporal and Real-Time Databases: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, *7*(4), 513–532. doi:10.1109/69.404027

Pacitti, E., Simon, E., & Melo, R. (1998). Improving data freshness in lazy master schemes. *Proceedings. 18th International Conference on Distributed Computing Systems*, 164-171.

Pandey, A. K., Pandey, S., & Shanker, U. (2019). LIFT- A New Linear Two-Phase Commit Protocol. *Proceedings of 25th annual International Conference on Advanced Computing and Communications (ADCOM 2019) at IIIT Bangalore.*

Pandey, S., Pandey, A., & Shanker, U. (2020). SP-LIFT: A Serial Parallel Linear and Fast-Paced Recovery-Centered Transaction Commit Protocol. *SN Computer Science, 1*(3).

Pandey, S., & Shanker, U. (2016). Transaction Execution in Distributed Real-Time Database Systems. *Proceedings of the International Conference on Innovations in information Embedded and Communication Systems*, 96-100.

Pandey, S., & Shanker, U. (2017). On Using Priority Inheritance Based Distributed Static Two Phase Locking Protocol. *Proceedings of the International Conference on Data and Information System (ICDIS)*, 179-188.

Pandey, S., & Shanker, U. (2018). A One Phase Priority Inheritance Commit Protocol. *Proceedings of the 14th International Conference on Distributed Computing and Information Technology (ICDCIT) Bhubaneshwar, India, January 11-13 2018*. doi:10.1007/978-3-319-72344-0_24

Pandey, S., & Shanker, U. (2018). IDRC: A Distributed Real-Time Commit Protocol. *Procedia Computer Science*, *125*, 290–296. doi:10.1016/j.procs.2017.12.039

Pandey, S., & Shanker, U. (2018). Priority Inversion in DRTDBS: Challenges and Resolutions. *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data (CoDS-COMAD '18)*, 305-309. doi:10.1145/3152494.3167976

Pandey, S., & Shanker, U. (2018). CART: A Real-Time Concurrency Control Protocol. In *22nd International Database Engineering & Applications Symposium (IDEAS 2018).* ACM.

Pandey, S., & Shanker, U. (2019). EDRC: An Early Data Lending based Real-Time Commit Protocol. Encyclopedia of Organizational Knowledge, Administration, and Technologies.

Pandey, S., & Shanker, U. (2019). MDTF: A Contention Aware Priority Assignment Policy for Cohorts in DRTDBS. In Encyclopedia of Organizational Knowledge, Administration, and Technologies, 1st Edition. IGI Global.

Pandey, S., & Shanker, U. (2020). CA-EQS: A Contention Aware Distributed Priority Assignment Heuristic. *The Journal of Supercomputing*.

Pandey, S., & Shanker, U. (2020). Causes, Effects, and Consequences of Priority Inversion in Transaction Processing. In *Handling Priority Inversion in Time-Constrained Distributed Databases*. IGI Global. doi:10.4018/978-1-7998-2491-6.ch001

Pandey, S., & Shanker, U. (2020). RACE: A Concurrency Control Protocol for Time-Constrained Transactions. *Arabian Journal for Science and Engineering*, *45*(12), 10131–10146. doi:10.1007/s13369-020-04632-1

Pandey, S., & Shanker, U. (2020). RAPID: A real time commit protocol. *Journal of King Saud University - Computer and Information Sciences*.

Pandey, S., & Shanker, U. (2020). Transaction Scheduling Protocols For Controlling Priority Inversion: A Review. *Computer Science Review*, *35*, 35. doi:10.1016/j.cosrev.2019.100215

Peddi, P., & DiPippo, L. C. (2002). A replication strategy for distributed real-time object-oriented databases. *Proceedings Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, 129-136. doi:10.21236/ADA477629

Pierce, D., Shepherd, S., & Johnson, D. (2019). Modelling the Impacts of Inter-City Connectivity on City Specialisation. *International Journal of System Dynamics Applications*, *8*(4), 47–70. doi:10.4018/IJSDA.2019100104

Said, A. H., Sadeg, B., Amanton, L., & l Ayeb, B. (2008). A Protocol to Control Replication in Distributed Real-Time Database Systems. *ICEIS,* (1), 501-504.

Said, A. H., Sadeg, B., Ayeb, B., & Amanton, L. (2009). The DLR-ORECOP real-time replication control protocol. *2009 IEEE Conference on Emerging Technologies & Factory Automation*, 1-8.

Salem, R., & Abdul-kader, H. (2016). Scalable data-oriented replication with flexible consistency in real-time data systems. *Data Science Journal*, 15.

Shanker, U., Misra, M., & Sarje, A. (2006). Some performance issues in distributed real-time database systems. *Proc. VLDB Ph.D. Work,Conv. Exhib. Cent. (COEX)*.

Shanker, U., Misra, M., & Sarje, A. K. (2006). SWIFT - A new real time commit protocol. *Distributed and Parallel Databases*, *20*(01), 29–56. doi:10.1007/s10619-006-8594-8

Singh, R. K., Pandey, S., & Shanker, U. (2019). A Non-Database Operations Aware Priority Ceiling Protocol for Hard Real-Time Database Systems. *10th International Conference on Computing Communication and Networking Technologies.* doi:10.1109/ICCCNT45670.2019.8944482

Son, S. H. (1988). Replicated data management in distributed database systems. *SIGMOD Record*, *17*(4), 62–69. doi:10.1145/61733.61738

Srivastava, A., Shankar, U., & Tiwari, S. K. (2012). A protocol for concurrency control in real-time replicated databases system. *International Journal of Computer Networks and Wireless Communications, 2*(3).

Srivastava, A., Shankar, U., & Tiwari, S. K. (2012). Transaction management in homogenous distributed real-time replicated database systems. *International Journal of Advanced Research in Computer Science and Software Engineering*, *2*(6).

Stankovic, J., Ramamritham, K., & Towsley, D. (1991). Scheduling in real-time transaction systems. *Foundations of Real-Time Computing: Scheduling and Resource Management*, 157-184.

Ulusoy, O. a. (1992). A simulation model for distributed real-time database systems. *25th Annual IEEE Proceedings Simulation Symposium*, 232-240. doi:10.1109/SIMSYM.1992.227558

Ulusoy, Ö. (1994). Processing real-time transactions in a replicated database system. *Distributed and Parallel Databases*, *2*(4), 405–436. doi:10.1007/BF01265321

Ulusoy, Ö. (1995). Research issues in real-time database systems. Survey paper. *Information Sciences*, *87*(1-3), 123–151. doi:10.1016/0020-0255(95)00130-1

Ulusoy, O., & Belford, G. (1992). Real-time lock-based concurrency control in distributed database systems. *Proceedings of the 12th IEEE International Conference on Distributed Computing Systems*, 136-143. doi:10.1109/ICDCS.1992.235046

Ulusoy, O., & Belford, G. G. (1992). Concurrency Control in Real-Time Database Systems. In *ACM annual conference on Communications* (pp. 181–188). ACM.

Ulusoy, Ö., & Belford, G. G. (1993). Real-time transaction scheduling in database systems. *Information Systems*, *18*(08), 559–580. doi:10.1016/0306-4379(93)90024-U

Wei, Y., Aslinger, A., Son, S., & Stankovic, J. (2004, August). ORDER: A dynamic replication algorithm for periodic transactions in distributed real-time databases. *10th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCS 2004)*.

Xiong, M., Ramamritham, K., Haritsa, J. R., & Stankovic, J. A. (2002). MIRROR: A state-conscious concurrency control protocol for replicated real-time databases. *Information Systems*, *27*(04), 277–297. doi:10.1016/S0306-4379(01)00053-9

Anupama Arun has received her B.Tech. degree in Computer Science and Engineering from Oriental Institute of Science and Technology (RGPV) Bhopal. She is currently pursuing M.Tech. from Madan Mohan Malaviya University of Technology, Gorakhpur, India.

Sarvesh Pandey, PhD., is presently working as an Assistant Professor of Computer Science at Banaras Hindu University, Varanasi, India. He received his Ph.D. degree (2020) in Computer Science & Engineering from M. M. M. University of Technology, Gorakhpur-273010, India. His broad areas of research include distributed real-time database systems, cloud computing and advanced data systems. He has published 01 book, 03 book chapters and more than 20 research papers in various international journals/conferences. He is also reviewer of some reputed journals, conferences, and book series.

Udai Shanker, PhD., is presently Professor in the Department of Computer Sc. & Engineering of M. M. M. University of Technology, Gorakhpur-273010. For his imitation of the most modern of approaches and also for his exemplary devotion to the field of teaching, and sharing his profound knowledge with students to make better future citizen of India, he has been a role model for the new generation of academicians. Besides introduced radical and revolutionary changes that have positively impacted the database world and student community, he is a man well versed with all the intricacies of academics.