Wireless Personal Communications (2023) 130:987–1004
https://doi.org/10.1007/s11277-023-10316-4

A Priority Inheritance Centered Locking Protocol for DRTDBS

Sarvesh Pandey[1] · **Udai Shanker**[2]

Accepted: 24 February 2023 / Published online: 11 March 2023

Abstract
In distributed real-time applications, the static 2 Phase Locking with High Priority (S2PL-HP) protocol may resolve data conflict(s) among transactions executing concurrently. S2PL-HP is virtuously free from the priority inversion that may arise due to executing–executing conflict—both the transactions in the execution phase. However, its performance may degrade because of other problems, i.e., cyclic restart and unnecessary abort, starvation of transactions of longer length, and system resource wastage. We propose a new Priority Inheritance (PRIN) centered locking protocol to resolve the above problems. PRIN checks the wastage of system resources by preventing cyclic restart and unnecessary abort using the priority inheritance approach. It is also free from deadlock and reduces the effects of lengthy transaction starvation. The system performance is measured using the transaction miss percentage metric. We developed the simulator to assess the performance of the proposed protocol; the results confirm that our system, PRIN, achieves 1.05×–1.23× performance improvement over the state-of-the-art systems.

Keywords Transaction · Two phase locking · DRTDBS · S2PL-HP · Priority inheritance

1 Introduction

Today, shared access to a database in a multi-user environment has been a key idea behind most computational decision-making systems. We use countless real-life database-powered applications today, ranging from manufacturing, payroll management, health care, banking, social media, robotics, satellite missions, agriculture systems, and many more. It is beyond doubt that all such applications are making our life easy. At the same time, with growing demand and exponentially increasing complexity of the systems, it has become even more challenging for today's database researchers to develop more efficient database handling algorithms and protocols to meet ever-growing computational demand [1, 2].

✉ Sarvesh Pandey
sarveshpandey@bhu.ac.in

Udai Shanker
udaigkp@gmail.com

[1] Computer Science – MMV, Banaras Hindu University, Varanasi, India

[2] Department of Computer Science and Engineering, M.M.M. University of Technology, Gorakhpur 273010, India

We access the database using transaction invocations [3]. The transaction is a logical grouping of queries involving database operations, and it must follow the ACID property: Atomicity, i.e., all or nothing; consistency, i.e., consistent data state at both points (before the beginning and after the end of the transaction execution life-cycle); isolation, i.e., invisible intermediate transaction state to users; durability, i.e., changes made by a successful transaction persist even in case of system failure [4].

Concurrent transaction execution has twofold benefits: improved system performance metrics (particularly throughput) and reduced wait time. However, concurrency is useless if achieved at the cost of losing database consistency. It is always about designing concurrency control protocols so that consistency is not compromised and performance improves. One way to ensure consistency is by implementing a lock manager. If concurrent access results in a lock conflict, the algorithm should have a tradeoff, like who would get access to the resource first. Application requirements—criticality, deadline [5, 6]—influence the decision to resolve lock incompatibility. Most of the commercial database-centered applications we use today utilize lock-based Concurrency Control (CC) mechanisms in scheduling the resources for the execution of the transactions. Two Phase locking protocol, i.e., 2PL—the first commercial concurrency control procedure—is utilized by conventional database systems that are centralized and do not consider the deadline.

S2PL-HP [7], 2PL-WP [8], and RT-S2PL [9] have the following limitations—priority inversion (PI), unnecessary aborts, wastage of system resources, lengthy transactions starvation, and cyclic restart (see Sect. 2 for details). PRIN is a cooperative cum competitive protocol that addresses the above limitations. First, it uses the priority inheritance mechanism to reduce resource wastage. Second, it allows the low-priority transaction to complete its execution provided it doe not cause a deadline miss for the waiting high-priority transaction. Third, it overcomes the frequent starvation associated with lengthy transactions.

Section 2, i.e., Related Work, sets a foundation for our proposed protocol. Section 3 describes the PRIN protocol. The evaluation of PRIN's performance is presented in Sect. 4; results confirm improvement with PRIN over S2PL-HP. Section 5 first concludes and then lists areas for future research work along the lines of this study.

2 Related Work

The 2PL is the sole of all the pessimistic CC mechanisms developed to facilitate concurrency. It works in two phases—growing and shrinking. The locking and unlocking of data items required by the transactions are performed in these two phases, respectively. Locking a data item is impossible during the shrinking phase, and unlocking cannot be done during the growing phase. Variants of 2PL are designed to meet the ever-changing database requirements of applications [9]. Two widely studied 2PL variants are static 2PL and dynamic 2PL. As the name suggests, with static 2PL protocol, all the locks are first acquired by a transaction, and then its execution is initiated. The requesting transaction would either lock all its prerequisite data items or not even a single. Though this feature leads to lower communication overhead in setting up the locks and eliminates the possibility of local deadlock, the same can be a reason for unbounded blocking of the requesting transaction. With dynamic 2PL, transaction dynamically requests data items as and when they need; it involves higher communication cost.

Let us take a scenario to understand the unbounded blocking problem with static 2PL. Suppose transaction T_1 has requested a lock on data items d_1 and d_2. Data item d_1 is

currently available, but some other transaction T_2 has already locked d_2. In this scenario, as per the static 2PL protocol, transaction T_1 will not put a lock on any of the above two data items. Meanwhile, transaction T_3 arrives and locks data item d_1, which is currently free; even when the earlier arrived transaction T_1 still requires access. Now, data item d_2 is released by T_2. So, even after the release of data item d_2, which was the reason for blocking of transaction T_1, T_1 cannot proceed for execution because now data item d_2 is not free. The above undesirable scenario, where the release of a conflicting data item(s) is not a guarantee that requesting transaction can further proceed for its execution, is unacceptable for the time-constrained databases [10–12].

Real-Time S2PL (abbreviated as RT-S2PL) protocol is specially designed for the distributed real-time database system (DRTDBS) environment [9]. The RT-S2PL is similar to the S2PL protocol as it also requires that all the prerequisite data are first locked, and then only a transaction may start its execution. Furthermore, it has a unique feature of assigning priority to all the data items accessed by concurrently executing transactions. In RT-S2PL, a data item's priority equals the highest requesting transaction's priority. With RT-S2PL, if a transaction cannot lock all its prerequisite data items at an instance because of some data conflict, it has to release any of the data items it has locked to ensure an 'all-or-none' property. Furthermore, when the priority of the data item is lower, the priority of the transaction requesting access to the data item is higher, and locking the data item is not possible, we would upgrade the data item's priority to that of requesting transaction. Thus, RT-S2PL overcomes the limitation with S2PL of prolonged blocking of the requesting transaction.

The 2PL Wait Promote protocol (2PL-WP) is the first locking-based protocol using the concept of priority inheritance. It resolves the PI problem by taking two actions—blocking the requesting transactions and inheriting their priority [7, 8]. Thus, the 2PL-WP protocol lessens the negative impact of the PI problem. Suppose a data conflict occurs between low-priority lock holder transactions and high-priority lock requester transactions. In that case, the low-priority lock holder transaction's priority is upgraded equally to the priority of the highest priority assigned to the transaction that has requested access for one of its locked data items so that it can complete its execution somewhat earlier. The priority up-gradation of low-priority lock holder transactions is to make the blocked high-priority transactions get the conflicting data item earlier than usual if the low-priority lock holding transaction had been executed at its original priority. In short, the above approach to dealing with the PI problem lessens the duration of the PI for high-priority lock explicitly requesting transactions instead of eliminating the PI problem [13–15]. The priority inheritance mechanism improves the commit protocols [16].

The static 2PL High Priority (S2PL-HP) protocol ensures that the transactions of low priority are not supposed to block high-priority transactions by resolving the PI problem immediately in favor of transactions of high priority [7, 8]. It is done by determining the issue of conflicts for the data instantaneously in favor of higher-priority transactions. The S2PL-HP protocol is not affected by the well-known destructive impact of PI in the RTDBS environment. However, in DRTDBS, the S2PL-HP protocol cannot eliminate PI. It is because the low priority lock holding transaction cannot be aborted if it is in a PREPARED state and is part of some global transaction.

RACE [17] protocol attempts to address five limitations of the existing locking protocols—the starvation of long transactions, deadlock, pseudo priority inversion, cyclic restart, and inefficient resource utilization. However, it is designed for a parallel transaction model; therefore, it does not suit when sub-transactions can independently run. An Enhanced Secure 2PL (ES2PL) [18] real-time protocol has recently been proposed, suggesting the role of a security level mechanism in resolving data conflicts. It is claimed

that ES2PL's throughput is incrementally better than RACE's with a lesser rollback count. Table 1 summarizes the key protocols discussed in this section and compares them with the proposed study.

From Table 1, it could be concluded that PRIN is a more robust concurrency control protocol for parallel DRTDBS systems. This study can be viewed as an extension of the heuristic presented in [15], which is the first in exploring the parallel execution of global transactions.

3 PRIN: A Real-Time Locking Protocol

The PRIN protocol is a lock-based concurrency control protocol designed for the DRTDBS environment. It solves the problems such as PI, unnecessary aborts, wastage of system resources, lengthy transaction starvation, and cyclic restart. Assume an example to explain the problem of cyclic restart from a practical point of view, as this problem was discussed a little in the context of time-constrained databases.

Let us assume that T_1 and T_2 are two global transactions with cohorts executing at multiple sites in parallel to complete their execution. Further in the timeline, a sub-transaction T_{11} of global transaction T_1 is granted all its prerequisite locks so that it may further start its processing. Note that it is assumed that the transaction execution is divided into phases, one being locking and processing as the second. Commit processing begins after it. The least slack first (LSF) policy is used for assigning priority to the transactions. Moreover, the priority of global transaction (T_1) is lower as compared to transaction T_2. At some later stage, a sub-transaction T_{2i} of global transaction T_2 requests to use by accessing one of the data which has already been utilized by sub-transaction T_{11}. Thus, the PI occurred, and as per the S2PL-HP protocol, the global transaction T_1 will be aborted along with all its participating cohorts. Such intermediate stage restart of global transaction T_1 will lead to the release of all its locked data items. This will pave the way for sub-transaction T_{2i} of global transaction T_2 to set a lock on the requested data item, which was previously held by sub-transaction T_{11} of global transaction T_1. Till this point, everything is as expected. But later, it was found that the restarting global transaction T_1 had already eaten up most of its slack time, so its priority is now assigned to be higher than that of global transaction T_2. After this, sub-transaction T_{11} of global transaction T_1 yet again requests access to the data items required for its execution. As previously said, some of its required data items

Table 1 A comparative comparison of similar protocols for concurrency control in a distributed real-time environment

Publication	Key comparison parameters for state-of-the-art systems			
	Unnecessary CRs, and PI	Deadlock	PseudoPI	Transaction model
2PL-HP [5]	✓	✓	✓	Serial
2PL-WP [7, 8]	✓	✓	✓	Serial
S2PL-HP [7, 8]	✓	✓	✓	Serial/parallel
RACE [17]	×	×	×	Serial
Priority heuristic [15]	✓	✓	✓	Parallel
PRIN (this study)	×	×	×	Parallel

CR means Cyclic Restart; PI means Priority Inversion; PseudoPI means Pseudo Priority Inversion

are now locked by sub-transaction T_{2i} of global transaction T_2. This situation again leads to the problem of PI. As per the S2PL-HP protocol, global transaction T_2 has now been restarted by T_1. Such cyclic restart eventually leads to the deadline miss of both global transactions. The above cyclic restart problem is a source of many other problems ranging from a waste of already consumed system resources, unnecessary abort, and degraded system performance.

The PRIN protocol resolves the above-discussed problems. The PRIN locking protocol aims to resolve the issue of data conflicts among concurrently executing transactions. Fulfilling it becomes even more difficult when transactions are associated with deadlines. Therefore, an efficient real-time conflict resolution procedure is being in need to improve the DRTDBS performance. For instance, let us assume a scenario where a cohort of low-priority distributed real-time transactions is currently holding a lock on some of the data items that are requested by some other cohort belonging to a high-priority distributed real-time transaction at a later stage. This scenario results in the problem of priority inversion that resulted only after considering real-time constraints of transactions. The simple approach to increasing the system throughput is by reducing the wastage of already consumed resources by handling their accesses intelligently. Like the conventional locking of data item-based concurrency control protocols, reducing the already consumed resource wastages is a goal in the case of real-time locking approach-based concurrency control protocol as well. However, the objective of real-time locking of the data item-based concurrency control protocols differs in a way that they primarily require that transactions complete their executions before their deadlines; increasing throughput and reducing resource wastages are secondary objectives.

The proposed PRIN protocol utilizes a new intermediate temporary priority assignment heuristic. This heuristic does not affect the initial priorities of transactions. While resolving conflict amongst concurrently executing transactions, this heuristic considers the cohort level priority as well. The role of the above intermediate priority assignment heuristic is explained with an example. Suppose a PI problem occurred because of data conflict among a lock holding low priority cohort (T_{Li}) and lock requesting higher priority cohort (T_{Hj}). Now, instead of simply aborting the T_{Li}, the heuristic decides the fate of T_{Li} based on the following parameters—(i). State of the T_{Li}, i.e., whether it is in a PREPARED state or not; (ii). The remaining execution time T_{Li} requires for completing its execution; and (iii) availability of slack time with the higher priority cohort T_{Hj}, which arrived after T_{Li}. Slack time is associated with every cohort at the time they initiate their execution; it is a reserved time available with a cohort in addition to its minimum execution time to deal with the uncertainty involved in its execution. The computation of the remaining execution time associated with the lock holding sub transaction T_{Li} is done as per the equation given below.

$$T_{Remain}\left(T_{Li}\right) = R_i - T_{Elapse} \tag{1}$$

where R_i is the minimum time required for the cohort response; T_{Elapse} is elapsed time for the execution of the T_{Li}; and T_{Remain} is the required remaining execution time of T_{Li}. By considering the still availability of slack time with the lock-requesting high-priority cohort, the PRIN protocol also handles the impact and the problem of the starvation of lengthy transactions that originate due to the higher level probability of conflicts for the accessing of the data items. The PRIN protocol reduces the transaction deadline miss percentage by considering the following three steps.

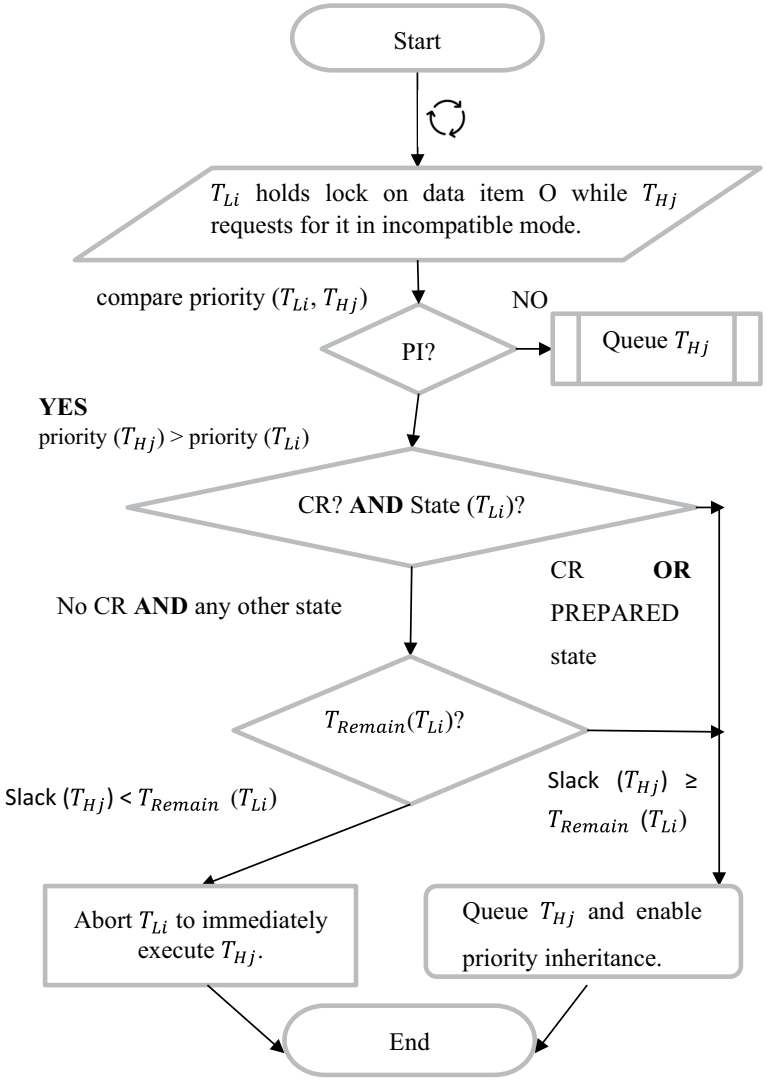

Fig. 1 Block diagram for data conflict resolution method in PRIN

1. If the lock-holding low-priority cohort T_{Li} is not in a PREPARED state and $T_{Remain}(T_{Li})$ is less than the slack time of (T_{Hj}), the priority of T_L inherits the priority of the T_H. Finally, T_H is queued in a wait state.
2. If lock-holding low-priority cohort T_{Li} is not in a PREPARED state and $T_{Remain}(T_{Li})$ is larger than or equitable to the slack time (T_{Hj}), low-level priority lock holder transaction is supposed to be aborted.
3. In the case of lock-holder low-priority cohort T_{Li} being in the state PREPARED, T_L inherits the priority of the T_H. The T_H is queued in a wait state irrespective of the requester transaction T_H being of higher priority or not. The priority inheritance procedure utilized here provides the transactions in a conflict state a fairer chance for successful completion by lessening the period of PI.

The block diagram is presented in Fig. 1 to understand better PRIN protocol working.

As can be seen from Fig. 1, if there is no priority inversion, the pre-defined process is triggered, which puts the requesting low-priority transaction in the wait queue. In the case of priority inversion, other conditions are evaluated to decide on the execution of one of the two possibilities. Let us present PRIN's data conflict resolution algorithm.

PRIN Algorithm

Input: T_{Hj} is a sub-transaction requesting to lock data item; T_{Li} is a sub-transaction already having a lock on data. The function priority (T_H) is the priority of T_H, and function priority (T_L^A) is the priority of T_L in case it got aborted in its previous attempt.

1. PRIN_lock_acquire ()
2. for each data item, D_i
3. **if** (! Conflict_for_lock)
4. Assign the data to T_{Hj};
   end if
5. Else
6. Examine the priority level of a conflicting cohort of T_{Li} having locked the data;
7. **if** (priority (T_{Hj}) > priority (T_{Li}))
8. **if** (priority (T_{Hj}) > priority (T_{Li}^A))
9. **if** (T_{Li} has not finished the processing yet)
10. **if** (slack time (T_{Hj}) >T_{Remain} (T_{Li})) //*slack time($T_{Hj} \geq 0$ *//
    insert T_{Hj} in wait queue;
    T_{Li} inherits priority of T_{Hj};
    end if
    else
    T_{Hj}aborts T_{Li};
    Allocate the intended data item to T_{Hj};
11. else
    queue T_{Hj} in wait state;
    T_{Li} inherits priority of T_{Hj};
    end if (match of line 9)
12. else
    T_{Hj}waits for the completion of T_{Hj};
    T_{Li} inherits priority of T_{Hj};
    end if (match of line 8)
13. else insert T_{Hj} in wait queue;
    end if (match of line 7)

In brief, optimistically PRIN protocol reduces the severe wastage of already consumed resources of the system by utilizing the priority inheritance procedure, eliminates the cyclic restart problem by prejudging its occurrence, and nicely overcomes the detrimental impact of the starvation associated with the transactions of long length up to a fairer way by utilizing the assigned intermediate temporary priority. The above protocol has shown how data locking is permitted in PRIN.

3.1 Key Contributions

The critical contributions made by the PRIN protocol are summarized below.

1. The PRIN protocol is free from local deadlock and reduces lengthy transaction starvation's adverse effects.
2. It reduces the wastage of system resources by preventing cyclic restarts and unnecessary abort using a priority inheritance mechanism.

4 Performance Evaluation

We developed the DRTDBS simulator to assess the fruitfulness of the PRIN protocol, with consideration of complete coverage of the performance setting acceptable to the industry/academic arena. The environment assumed for this study is quite similar to the performance settings of past studies [19–21] on transaction processing in DRTDBS.

4.1 The DRTDBS Simulation Model

The simulator is written in C language to assess the PRIN protocol's performance [22, 23]. The discrete event-based programming style is utilized. In each simulator run, 20 thousand transactions are invoked; an average of 10 independent runs are taken to conclude any behavior. Such repeated experiments are the foundation of confidence in insights drawn. Each transaction adds equal value to the system; no transaction is more critical than the other [24]. Figure 2 presents the high-level view of the DRTDBS model implemented, facilitating the parallel execution of firm-distributed transactions.

We particularly schedule two types of resources—data and CPU. To schedule CPU, a priority-based preemption mechanism is implemented; data is provided to high-priority transactions first on a preemptive basis if some prepared cohort does not hold it. Furthermore, the assumptions considered while developing the model are given below.

- Poisson distribution-based transaction arrival is followed. The firm-distributed transactions are considered, where deadline miss leads to transaction kill.
- A transaction is just a logical combination of read/write data operations. All the data items and their access mode are first recorded, and then transaction execution begins.
- Communication between any two shards in the global system will cost either 10 ms or 1 ms. This helps simulate network delay's effects on overall system performance indicators.
- The cohort-level execution is done in parallel, as there is no data dependency between sibling cohorts of any global transaction. However, to facilitate communication between sibling cohorts, each cohort is provided with the IDs of all its sibling cohorts.

The performance of the PRIN is assessed using the transaction miss percent metric (TMP); the purpose is to reduce TMP. Mathematically, one can compute TMP using the below equation,

$$\text{TMP (in \%)} = \left(\frac{\text{\# transaction abort count}}{\text{\# total transaction count}}\right) * 100 \tag{2}$$

TMP signifies aborted transactions percentage. To avoid any biasness in the final result, the system load has been varied during experimentation—normal (TMP ranges from 0 to 20), heavy (TMP ranges from 20 to 100), and mix load (TMP ranges from 0 to 100) [7].

4.2 Deadline Computation in a Parallel Setting

The mathematical foundation of the parallel DRTDBS model is discussed in this subsection. The deadline of any transaction depends on three parameters—arrival time, execution time, and slack factor. Slack factor plays a significant role in serial DRTDBS setting [22]; we employed the CA-EQS policy for priority assignment [25]. The ordering-based deadlock resolution in CA-EQS works well for global transactions as well. In a parallel setting, the computation of predicted execution time (PET) is performed differently compared to the serial DRTDBS. The PET further depends on two sub-components—execution and commit time. We focus on execution time ($time_{exe}$) for this study, deatails on computation of commit time ($time_{commit}$) can be found in [26].

$$\text{PET} = time_{exe} + time_{commit} \tag{3}$$

The execution time of a cohort (sub-transaction) can be computed as a multiple of the number of operations it performs. To begin with, let us calculate the time required to complete a single data operation ($time_{\text{ONE}}$),

$$time_{ONE} = time_{\text{lock}} + \left(\text{time}_{\text{w}} \times \text{probablility}_{\text{w}}\right) + \left(\text{time}_{\text{r}} \times \text{probability}_{\text{r}}\right) \tag{4}$$

Here, time_{r} and time_{w} are a time to read and write respectively. The $\text{probablility}_{\text{r}}$ and $\text{probablility}_{\text{w}}$ are the probability of read and write operations, respectively. As can be seen from Eq. 4, performing a data operation is a two-step process – lock the data item

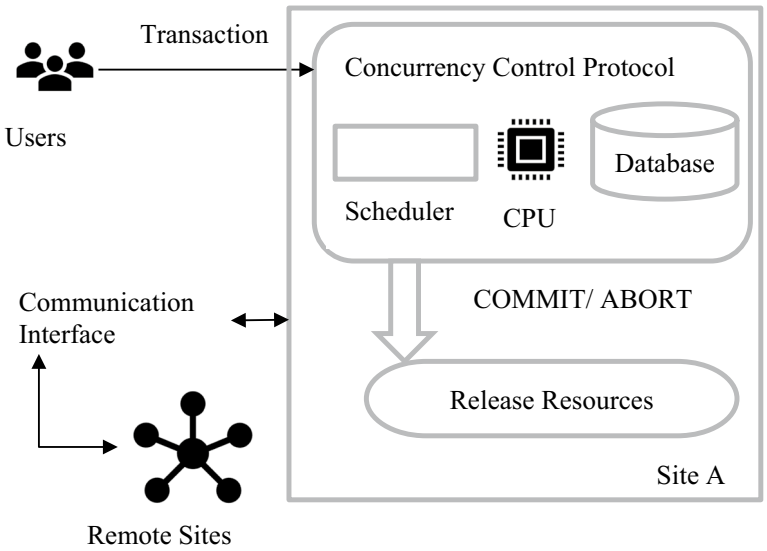

Fig. 2 Parallel transaction execution model for DRTDBS

in shared/exclusive mode and then do computation. Assuming a local transaction with 'n' operations, the local execution time ($time_{exe\_local}$) is computed as,

$$time_{exe\_local} = time_{ONE} \times n \tag{5}$$

Execution of global transactions, which consists of multiple cohorts, can be optimized in parallel DRTDBS. Unlike executing a local transaction, it requires message exchange because of having more than one cohort. Moreover, the total number of data operations (n) is the sum of local (n_{local}) and remote (n_{remote}) operations.

$$n = n_{local} + n_{remote} \tag{6}$$

Furthermore, the execution time for a global transaction, considering the parallel execution of cohorts, can be computed as,

$$time_{exe\_global} = MAX\left(time_{ONE} \times n_{local}, MAX\{\text{cohort\_exe}\}_{i=1}^{r}\right) + comm\_cost \tag{7}$$

Here, 'cohort_exe' is the time required for executing a sub-transaction at some cohort; 'r' is the cohort count; '*comm_cost*' is the message count for running a global transaction. From Eq. 7, it could be interpreted that parallelly running cohorts reduces the duration of a global transaction.

4.3 Parameters for Experimentation

The database populated for our experimentation has 2400 pages per shard. There is a total of 4 non-replicated shards, which means data present at geographically distinct sites are distinct. The reason for going with a tiny database is to create high data contention. The higher the data contention, the higher will be PI probability. This is the best environment to assess the effectiveness of the proposed protocol in handling the PI problem [22]; the default simulation parameters considered in our study are listed in Table 2.

The default set of parameters is primarily utilized for conducting experiments unless specified otherwise. Any change in the parameter setting may affect the final result; however, the qualitative trend is expected to remain the same. A general experimentation framework is designed to add confidence to the qualitative findings. At the same time, variation of data and resource contention also led to some interesting results. To quantitatively assess the contention level, metrics like CPU utilization, disk utilization, and data conflict frequency are used.

4.4 Experiment Findings

The experiments are designed in two categories, considering whether the database resided in primary or secondary memory (main memory resident or disk-resident). Moreover, the PRIN is evaluated against the S2PL-HP protocol as it is the improved version of it.

4.4.1 Experiment 1 Using Main Memory Databases

The idea of making the entire database resident in main memory comes from the recent advancements in computer storage facilities. Today, having a main memory size of more

than 100 GB is becoming common, particularly in an industry setting. This has become possible because of less cost in gaining such capacity with declining hardware cost. Its well-known data access fastens multifold with main memory compared to hard disk. The main memory resident database is evaluated as one of the performance-driving aspects.

Figure 3 evaluates the TMP of S2PL-HP and PRIN protocol under 'normal & heavy' load conditions. Since the database resides in the main memory and the communication delay is 1 ms, the TMP is considerably low for both protocols. As expected, the PRIN protocol performs incrementally well as compared to S2PL-HP. It supports the argument that low communication delay leads to considerably shorter transaction completion time, and therefore there is a lesser chance of cyclic restart among transactions. The performance improvement is primarily due to the intelligent use of a priority inheritance technique.

Figure 4 evaluates the TMP of S2PL-HP and PRIN protocol under 'normal & heavy' load. Here, the TMP is very high for both protocols compared to Fig. 3. This is due to an increase in overall transaction execution time with an increase in communication delay from 1 to 10 ms, which resulted in more transactions missing their deadlines. As expected, the PRIN performs significantly better than S2PL-HP due to the elimination of cyclic restart. It supports the argument that a more significant communication delay leads to considerably larger transaction completion time and, therefore, more chances of transaction deadline miss due to cyclic restart. The performance improvement is primarily due to eliminating the cyclic restart problem in PRIN; intelligent use of the priority inheritance technique also resulted in improved system performance.

4.4.2 Experiment 2 Using Disk Resident Databases

Figures 5, 6 and 7 show the TMP as a function of system workload. The system misses more deadlines with an increase in workload, resulting from long queuing delays, a high probability of conflict for the data, and a higher probability of blocking transactions.

Figure 5 evaluates the TMP of S2PL-HP and PRIN protocol under normal load. Since the communication delay is 1 ms and there is low data contention, the deadline miss percentage of the transaction is considerably lower for both protocols. The PRIN protocol can also be expected to perform additively well compared to the S2PL-HP. It supports the argument that low communication delay leads to considerably shorter transaction completion time and, therefore, lesser chances of cyclic restart among transactions. The performance improvement is primarily due to the intelligent use of a priority inheritance technique.

Table 2 System parameter and data settings

Parameter	Description and value
DB_{Size}	2400 pages per shard
N_{db}	4 shards
AR	[0,4] or up to four transactions per sec (distribution is uniform)
T_{com}	Either 1 ms or 10 ms of comm. delay
N_{op}	4–20 operations per transaction (distribution is uniform)
SF	Slack factor of 1–4 (distribution is uniform)
P(w)	0.6 is the probability of the write operation
CPU_{page}	5 ms to access a CPU page
$Disk_{page}$	20 ms to access a disk page

Figure 6 evaluates the TMP of S2PL-HP and PRIN protocol under heavy load. Since data contention is high, the TMP is at a high level for both the protocols mentioned above compared in Fig. 5. The PRIN again performs incrementally well compared to S2PL-HP. It supports the argument that lower-level communication delay leads to considerably shorter transaction completion time, and therefore, there are fewer chances of cyclic restart among transactions. The performance improvement is primarily due to the intelligent use of a priority inheritance technique.

Figure 7 evaluates the TMP of S2PL-HP and PRIN protocol under 'normal & heavy' load. Here, the TMP is very high for both protocols compared to Figs. 5 and 6. This is due to an increase in overall transaction execution time with an increase in communication delay from 1 to 10 ms, which results in more transactions missing their deadlines. As expected, the PRIN performs significantly better than S2PL-HP due to the elimination of cyclic restart. It supports the argument that a larger communication delay leads to a considerably larger transaction completion time, and therefore there are more chances of transaction miss due to cyclic restart. In addition to the intelligent use of the priority inheritance technique, the performance improvement is primarily due to eliminating the cyclic restart problem in the PRIN protocol.

The study of Figs. 4 and 7 shows that the TMP is usually higher for disk-resident databases than the main memory databases. Per our intuition, disk-resident databases require a larger transaction execution time since the data items residing on the disk need to be swapped to the main memory for processing. Such overhead is avoided in main memory-resident databases to minimize the transaction execution time.

Figures 3, 4, 5, 6 and 7 shows that the proposed PRIN protocol performs significantly better than the S2PL-HP protocol, irrespective of the type of load (normal, heavy, and mixed). Use of priority inheritance technique (instead of aborting lock holding low priority cohort) when a lock holding low priority cohort has not sent a PREPARED message to its coordinator and a lock requesting high priority cohort has a slack time greater than the remaining execution time of a low priority lock holding transaction, to give a fair chance

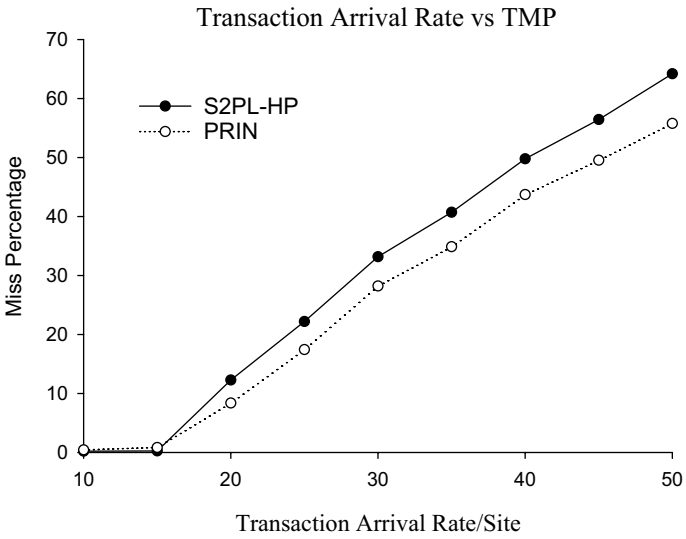

Fig. 3 Arrival rate versus TMP: Comm. Delay = 1 ms; load = mix; resource and data contention

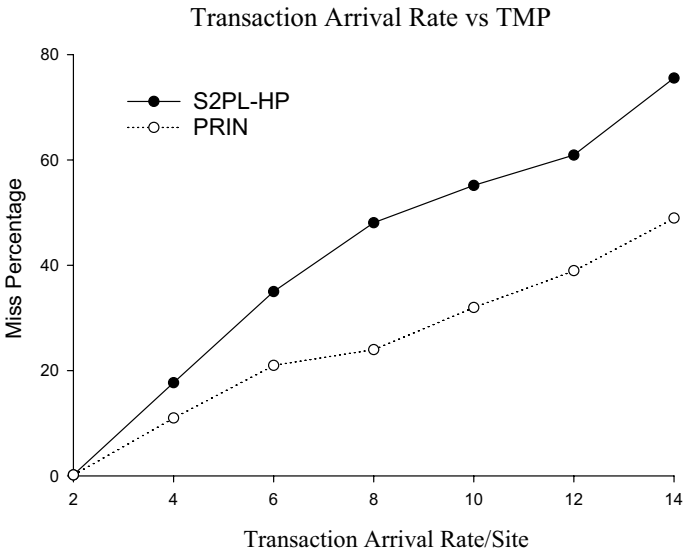

Fig. 4 Arrival rate versus TMP: Comm. Delay = 10 ms; load = mix; resource and data contention

of completion to the lock holding low priority cohort involved in a conflict. Such action results in the avoidance of the unnecessary abort of lock holding low priority cohort, declination in starvation with lengthy transactions, and lead to the fair utilization of resources. In addition to the above reasons, eliminating the cyclic restart problem also plays an important role in the performance improvement of the PRIN protocol.

5 Conclusions and Future Research

Existing real-time locking protocols suffer badly due to issues, i.e., cyclic restart, unnecessary abort, starvation of transactions of extended length, and wastage of the already consumed system resources. Resolving the above problems required developing a more efficient real-time locking protocol. The proposed PRIN protocol reduced the wastage of system resources by preventing cyclic restart and unnecessary abort using a priority inheritance mechanism. It is also deadlock-free and declines the adverse effects of lengthy transaction starvation. The PRIN protocol has shown better performance over the S2PL-HP by optimally utilizing the priority inheritance mechanism and combining the initial and intermediate strategies of priority assignments. We designed the DRTDBS simulator to compare the PRIN protocol's performance with the S2PL-HP protocol, and the simulation results confirmed the performance improvement. From the experiments, two major conclusions are drawn. First, an in-memory database is recommended if there is no financial constraint, as PRIN performed better with main-memory databases. Second, under higher communication delay, the performance benefits of PRIN got 2 ×; this is mainly because of better data conflict resolution.

Future time-constrained databases should be able to efficiently process an enormously varying number of transactions executing concurrently [12]. We discuss below selected areas of research with enormous possibilities.

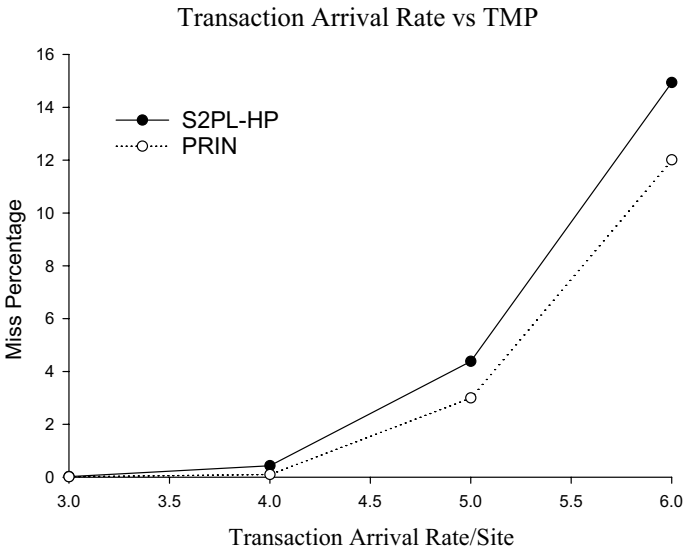

Fig. 5 Arrival rate versus TMP: Comm. Delay = 1 ms; Load = normal; resource and data contention

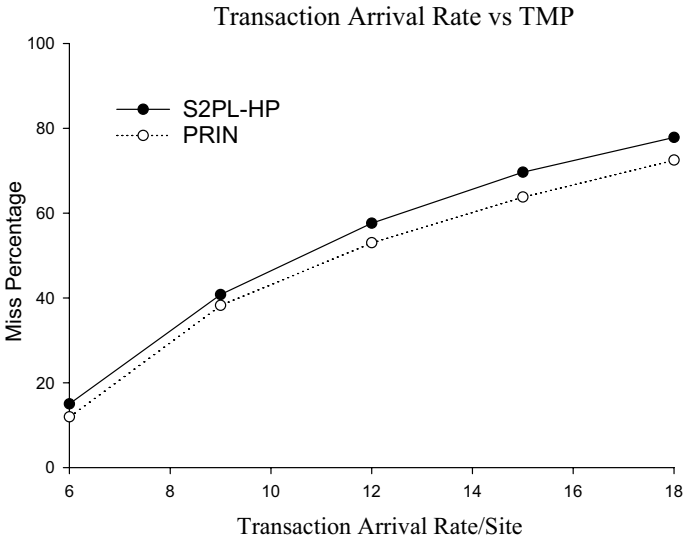

Fig. 6 Arrival rate versus TMP: Comm. Delay = 1 ms; load = heavy; resource and data contention

1. We can manage the priority inversion—originating from the executing-committing conflict because of the concurrent execution of distributed RTTs—by exploiting either the priority inheritance mechanism [16, 25, 26] or the lender-borrower approach [27–30], or both in a combined way [31]. These approaches have their positives and negatives [32]. More research is needed to develop advanced priority assignment heuristics for DRTDBS [33, 34].

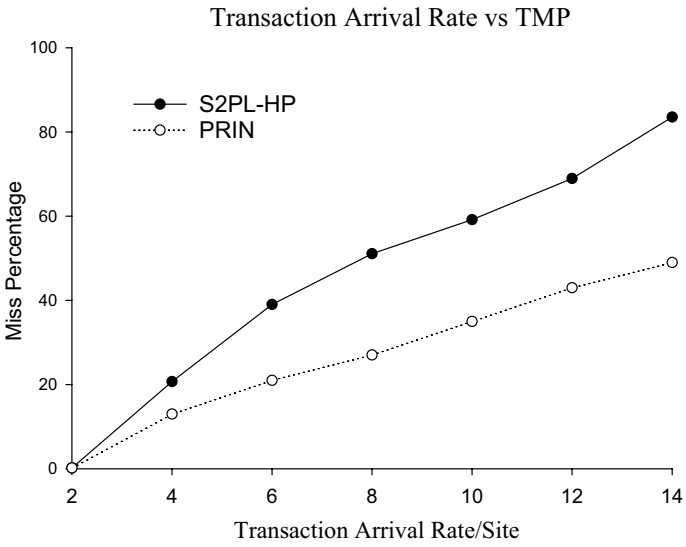

Fig. 7 Arrival rate versus TMP: Comm. Delay = 10 ms; load = mix; resource and data contention

2. After almost 3 decades of research in the conventional database domain, we need a reverse engineering approach to modify/redesign the foundational components of databases, e.g., 2PC and 2PL [35–38]. Furthermore, it is interesting to draw attention to the design of some applications that might involve database and non-database operations to achieve their objectives [39].
3. It will be noteworthy to extend the discussion on the PI problem in Mobile DRTDBS [40–42], and Replicated DRTDBS [43–45]. The study presented can also be extended to wireless sensor networks [46–48] as the backbone remains the same, which is a distributed system.
4. We plan to explore the intersection of blockchain and distributed databases to move to a decentralized data management setting [49, 50].

Acknowledgements The financial assistance under the Institute of Eminence (IoE) seed grant by BHU, Varanasi, India, is acknowledged.

Data Availability My manuscript has no associated data as the experiment-specific data is generated randomly through simulation.

References

1. Sadoghi, M., & Blanas, S. (2019). Transaction processing on modern hardware. *Synthesis Lectures on Data Management, 14*(2), 1–138.
2. 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.
3. Pandey, S., & Shanker, U. (2016). Transaction execution in distributed real-time database systems. In *Proceedings of the international conference on innovations in information embedded and communication systems* (pp. 96–100).

4. Pandey, S., & Shanker, U. (2020). Transaction scheduling protocols for controlling priority inversion: A review. *Computer Science Review, 35*, 100215.
5. Kao, B., & Garcia-Molina, H. (1993). An overview of real-time database systems. *Real Time Computing, 127*, 261–282.
6. Lam, K. Y. (1994). Concurrency control in distributed real time database systems. Ph.D. thesis.
7. Haritsa, J. R., Carey, M. J., & Livny, M. (1992). Data access scheduling in firm real-time database systems. *Real-Time Systems, 04*(03), 203–241.
8. Abbott, R. K., & Molina, H. G. (1992). Scheduling real-time transactions: A performance evaluation. *ACM Transactions on Database Systems, 17*(03), 513–560.
9. Lam, K.-Y., Hung, S.-L., & Son, S. H. (1997). On using real-time static locking protocols for distributed real-time databases. *Real-Time System, 13*(02), 141–166.
10. Yu, P. S., Wu, K.-L., Lin, K.-J., & Son, S. H. (1994). On real-time databases: Concurrency control and scheduling. *Proceedings of the IEEE, 82*(01), 140–157.
11. Ramamritham, K. (1993). Real-time databases. *Distributed and Parallel Databases, 01*(02), 199–226.
12. Shanker, U., Misra, M., & Sarje, A. K. (2008). Distributed real time database systems: Background and literature review. *International Journal of Distributed and Parallel Databases, Springer Verlag, 23*(02), 127–149.
13. Huang, J., Stankovic, J. A., & Towsley, D. (1991). On using priority inheritance in real-time databases. In *Real-time systems symposium* (pp. 210–221).
14. Huang, J., Stankovic, J. A., Ramamritham, K., Towsley, D., & Purimetla, B. (1992). Priority inheritance in soft real-time databases. *Real-Time Systems, 04*(03), 243–278.
15. Shanker, U., Misra, M., & Sarje, A. K. (2005). Priority assignment heuristic to cohorts executing in parallel. In Proceedings of the 9th WSEAS international conference on computers, World Scientific and Engineering Academy and Society (WSEAS) (pp. 01–06).
16. Pandey, S., & Shanker, U. (2018). A one phase priority inheritance commit protocol. In *Proceedings of the 14th international conference on distributed computing and information technology (ICDCIT) Bhubaneshwar, India, January 11–13, 2018*.
17. Pandey, S., & Shanker, U. (2020). RACE: A concurrency control protocol for time-constrained transactions. *Arabian Journal for Science and Engineering, 45*, 10131–10146.
18. Abduljalil, E., Thabit, F., Can, O., Patil, P. R., & Thorat, S. B. (2022). A new secure 2PL real-time concurrency control algorithm (ES2PL). *International Journal of Intelligent Networks, 3*, 48–57.
19. Ulusoy, O. (1995). A study of two transaction-processing architectures for distributed real-time data base systems. *The Journal of Systems and Software, 31*(02), 97–108.
20. Taina, J., & Son, S. H. (1999). Towards a general real-time database simulator software library. *IFAC Proceedings, 32*(01), 75–80.
21. Ulusoy, Ö., & Belford, G. G. (1993). Real-time transaction scheduling in database systems. *Information Systems, 18*(08), 559–580.
22. Lee, V. C. S., Lam, K.-W., & Hung, S.-L. (2002). Concurrency control for mixed transactions in real-time databases. *IEEE Transactions on Computers, 51*(7), 821–834.
23. Shanker, U., Misra, M., & Sarje, A. K. (2006). SWIFT—A new real time commit protocol. *Distributed and Parallel Databases, 20*(01), 29–56.
24. Stankovic, J., & Zhao, W. (1988). On real-time transactions. *ACM Sigmod Record, 17*(1), 4–18.
25. Pandey, S., & Shanker, U. (2017). On using priority inheritance based distributed static two phase locking protocol. In *Proceedings of the international conference on data and information system (ICDIS)* (pp. 179–188).
26. Pandey, S., & Shanker, U. (2018). CART: A real-time concurrency control protocol. In B. C. Desai, J. Hong, & R. McClatchey (Eds.), *22nd International database engineering & applications symposium (IDEAS 2018)*. New York: ACM.
27. Shanker, U., Agarwal, N., Tiwari, S., Goel, P., & Srivastava, P. (2010). ACTIVE-a real time commit protocol. *Wireless Sensor Network, 2*(3), 254.
28. Shanker, U., Vidyareddi, B., & Shukla, A. (2012). PERDURABLE: A real time commit protocol. In *Recent trends in information reuse and integration* (pp. 1–17).
29. Pandey, S., & Shanker, U. (2017). IDRC: A distributed real-time commit protocol. *Procedia Computer Science, 125*, 290–296.
30. Pandey, S., & Shanker, U. (2019). EDRC: An early data lending based real-time commit protocol. In *Encyclopedia of organizational knowledge, administration, and technologies* (1st Edn).
31. Pandey, S., & Shanker, U. (2018). Priority inversion in DRTDBS: challenges and resolutions. In *Proceedings of the ACM India joint international conference on data science and management of data (CoDS-COMAD' 18)* (pp. 305–309).

32. Haritsa, J. R., Ramamritham, K., & Gupta, R. (2000). The PROMPT real-time commit protocol. *IEEE Transactions on Parallel and Distributed Systems, 11*(02), 160–181.
33. Shanker, U., Misra, M., & Sarje, A. (2001). Hard real time distributed database systems: Future directions. In *Proceedings of all India seminar on recent trends in computer communication networks.* Dept. of ECE, IIT Roorkee, India (pp. 172–177).
34. Pandey, S., & Shanker, U. (2019). MDTF: A contention aware priority assignment policy for cohorts in DRTDBS. In D. R. M. Mehdi Khosrow-Pour (Ed.), *Encyclopedia of organizational knowledge, administration, and technologies* (1st Edn.).
35. Gupta, S., & Sadoghi, M. (2018). "EasyCommit: A non-blocking two-phase commit protocol. In *International conference on extending database technology (EDBT)* (pp. 157–168).
36. Gupta, S., & Sadoghi, M. (2019). Efficient and non-blocking agreement protocols. *Distributed and Parallel Databases, 38*, 1–47.
37. Pandey, A. K., Pandey, S., & Shanker, U. (2019). LIFT—A new linear two-phase commit protocol. In *Proceedings of 25th annual international conference on advanced computing and communications (ADCOM 2019) at IIIT Bangalore*.
38. Harding, R., Aken, D. V., Pavlo, A., & Stonebraker, M. (2016). An evaluation of distributed concurrency control. *VLDB, 10*(05), 553–564.
39. Singh, R. K., Pandey, S., & Shanker, U. (2019). A non-database operations aware priority ceiling protocol for hard real-time database systems. In *The proceedings of 10th international conference on computing communication and networking technologies*, IIT, Kanpur, India, July 6–8.
40. Lam, K., Kuo, T., Tsang, W., & Law, G. (2000). Concurrency control in mobile distributed real-time database systems. *Information Systems, 25*(4), 261–286.
41. Lei, X., Zhao, Y., Chen, S., & Yuan, X. (2009). Concurrency control in mobile distributed real-time database systems. *Journal of Parallel and Distributed Computing, 69*(10), 866–876.
42. Swaroop, V., Gupta, G. K., & Shanker, U. (2011). Issues in mobile distributed real time databases: Performance and review. *International Journal of Engineering Science and Technology, 3*(4), 3504–3517.
43. 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.
44. Wei, Y., Aslinger, A., Son, S., & Stankovic, J. (2004). ORDER: A dynamic replication algorithm for periodic transactions in distributed real-time databases. *In 10th international conference on real-time and embedded computing systems and applications (RTCS 2004)*, August.
45. 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 Engeering, 2*(6), 190–196.
46. Ashraf, S. (2019). Culminate coverage for sensor network through bodacious-instance mechanism. *i-Manager's Journal on Wireless Communication Networks, 8*(3), 1–9.
47. Ashraf, S., Alfandi, O., Ahmad, A., Khattak, A. M., Hayat, B., Kim, K. H., & Ullah, A. (2020). Bodacious-instance coverage mechanism for wireless sensor network. *Wireless Communications and Mobile Computing, 2020*, 1–11.
48. Ashraf, S., Ahmed, T., & Saleem, S. (2021). NRSM: Node redeployment shrewd mechanism for wireless sensor network. *Iran Journal of Computer Science, 4*, 171–183.
49. McConaghy, T., Marques, R., Müller, A., De Jonghe, D., McConaghy, T., McMullen, G., Henderson, R., Bellemare, S., & Granzotto, A. (2016). Bigchaindb: A scalable blockchain database. *White Paper, BigChainDB.*
50. Peng, Y., Du, M., Li, F., Cheng, R., & Song, D. (2020). FalconDB: Blockchain-based collaborative database. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data* (pp. 637–652).

Dr. Sarvesh Pandey is presently working as Assistant Professor in Computer Science—MMV, BHU, Varanasi, India. He received his Ph.D. degree (2020) in Computer Science and Engineering from M. M. M. University of Technology, Gorakhpur, India. His broad research areas include distributed real-time database systems, cloud computing, blockchain, and advanced data systems. More than 25 research papers have been published by him in various journals/conferences of their reputes. He serves as an active review member for various reputed journals/conferences/book series.

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