Contents lists available at ScienceDirect

Computer Science Review

journal homepage: www.elsevier.com/locate/cosrev

Transaction scheduling protocols for controlling priority inversion: A review

Sarvesh Pandey [*], Udai Shanker

Department of CSE, M. M. M. University of Technology, Gorakhpur 273010, India

ARTICLE INFO

Article history:
Received 26 May 2019
Received in revised form 24 September 2019
Accepted 12 November 2019
Available online 19 November 2019

Keywords:
Commit protocol
Concurrency control
Execute–execute conflicts
Execute commit conflicts
Priority inversion
DRTDBS

ABSTRACT

In advanced real-time distributed computing databases, the main performance criterion is to reduce the 'deadline miss' by the transactions; of course, consistency constraints also need to be satisfied. The goal of these applications is not to provide simply real-time transaction execution, but rather to provide a highly predictable, analysable, schedulable and reliable distributed computing platform to the users. The problem of resource conflicts amongst distributed real-time transactions and their handlings through various priority scheduling protocols highly affect the performance of the underlying applications. The past research works were mostly restricted to extend the traditional transaction processing techniques to resolve the issue of conflicts, and thus to improve the performance. The last review paper, largely on this issue, appeared in Shanker et al. (2008) [1]; since then many noteworthy algorithms have been described in the literature. Till date, no study was found discussing transaction processing techniques with data conflict issue in focus. Hence, our objective is to comprehensively discuss the state-of-the-art transaction scheduling protocols with an emphasis on the handling of execute–execute & execute–commit conflicts, and real-time optimistic concurrency control (OCC) protocols. The strengths and weaknesses of existing approaches are also discussed.

© 2019 Elsevier Inc. All rights reserved.

Contents

1. Introduction ... 1
2. Fundamentals of transaction processing in time constrained databases ... 2
3. Execute–execute conflict with pessimistic concurrency control protocols ... 5
4. Execute–commit conflict with pessimistic concurrency control protocols ... 6
5. Data conflict with optimistic concurrency control protocols ... 9
6. Scope for future research ... 10
7. Conclusions ... 12
Declaration of competing interest ... 12
Acknowledgment ... 12
References ... 12

1. Introduction

Databases have become an inseparable part of modern and fast-growing societies. In a day, at large, we work over several activities involving certain relations with databases [2]. Some of the extensively popular database applications are performing DEBIT-CREDIT operations at a bank; booking room in a hotel; reservation of bus, rail or airline tickets; searching for a bibliographic item from digital libraries; and using e-commerce sites for purchasing most of the things that are deeply connected with our daily needs [3]. In fact, database system (DBS) consists of the concrete physical database along with a database management system (DBMS) as an interface to interact; the concrete physical database may be defined as a collection of logically interrelated data items in an organized fashion to be shared by a large number of users [4]. Any user can interact with the database using "partially ordered set of read and/or write actions" named as a transaction. The database guarantees to support ACID principle – Atomicity, Consistency, Isolation, and Durability – for the execution of a transaction [5]. When data becomes voluminous, it may become increasingly difficult/inefficient to keep the entire data in a solely single database. In any such case, it may be more

* Corresponding author.
E-mail addresses: pandeysarvesh100@gmail.com (S. Pandey), udaigkp@gmail.com (U. Shanker).

https://doi.org/10.1016/j.cosrev.2019.100215
1574-0137/© 2019 Elsevier Inc. All rights reserved.

efficient to have a distributed database system (DDBS) consisting of concrete physical databases connected through a network, and a distributed DBMS [6]. The DDBS provides a means to let users feel that they are interacting with a single database even when they are accessing a data distributed over geographically distant locations.

There are multidimensional research aspects of the priority inversion problem. This problem can occur in any system/ application that uses the concept of priority in scheduling its resources. It is a situation where task having high priority is being forcefully put-off until the low priority task's final execution; the conflicting resource is released only after the completion of low priority task. The possibility of priority inversion entangles the analysis of systems that use priority-based schedulers since it invalidates the assumption that a task can be delayed only by higher-priority tasks [7,8]. At first, the priority inversion problem was introduced in the context of priority scheduling and monitors in [9]. Common sources of priority inversion are semaphores and critical sections, Ada tasking [10], software queues, and hardware queues [11]. The negative effect of priority inversion due to such priority inversion sources can be reduced using techniques mainly selective disable task preemption, and hardware priority queue with overwrite. Later, the priority inversion problem was widely discussed in the context of real-time operating systems (RTOS) [12]. With the help of performance studies, it has been shown that priority inheritance approach gives notable performance improvement in the RTOS. A study about benchmarking of the RTOS was presented in [13] on the basis of rhealstone. Two RTOS, Freescale MQX, and Quadros RTXC were benchmarked. The results have shown that there can be a substantial difference in small RTOS performance.

Real-Time Systems (RTS) require algorithms that can ensure that results are produced within a specified deadline [14,15]. L. Sha et al. [16] studied the problem of priority inversion in RTS and proposed a priority inheritance approach to overcome this problem to a limited degree. Here, priority inversion is formally defined, and as a solution to it, a new priority inheritance protocol was proposed and claimed to resolve the unbounded delay resulting due to priority inversion. Real-time database systems (RTDBS) are often used as a component of the RTS since the requirements of timelessness cannot be met by the traditional databases [17–19]. The RTDBS are often dispensed over a set of independent sites because of the inherently distributed nature of many applications having real-time requirements. This resulted in a requirement of a distributed real-time database systems (DRTDBS) which can be perceived as a join of positive features of both RTS and DDBS [20]. Hence, DRTDBS is a special type of DDBS where transactions, clearly, have timing constraints also named as deadlines. The deadline of a transaction is defined as a certain time in the future by which a transaction's execution desired to be finished. Task (a logical unit of work of RTS) and transaction (a logical unit of work of DRTDBS) are apparently similar. But, they (task and transaction) are entirely different notions, and therefore require different algorithms for their handling. Tasks can be preempted, and they may violate consistency [21]. On the contrary, transactions cannot always be preempted and they must ensure consistency.

While considering the consequences of deadline misses, real-time transactions (RTTs) can be classified as soft, firm or hard RTTs [22,23]. In the case of soft RTTs, a design aim is to reduce average response time and yielding a late result can be within acceptable limits [24]. However, deadline violation leads to the proportional decrement in the value of the result. Most of the audio-video streaming applications are based on a distributed soft RTS. Here, synchronization between the two flows (audio and video) is important for the overall quality. In the case of firm RTT, if the deadline expires then it (firm RTT) will be instantly aborted, and all the resources possessed by it get freed. A weather forecast system (if an extreme weather condition is predicted before its arrival, then the forecast system has successfully done its job. Prediction after the extreme event has already happened, or when it is happening is of no value) uses firm deadline RTTs. With hard RTTs, there is a negative value on coming up with a late result. The negative value represents that a catastrophic consequence may occur on deadline miss of a hard RTT [25]. The hard RTDBS often involve typical safety-critical control activities with hard deadline transactions where unpredictability can be extremely hazardous. Some well-known hard real-time applications are nuclear power plants, space shuttle avionics system, automatic weapon firing system, and pacemakers (in medical). Fig. 1 represents the type of transactions based on the impact of their deadlines on their associated values.

One must take into consideration the type of transaction like whether it is soft, firm or hard while designing the transaction scheduling algorithms to resolve the priority inversion problem. For any distributed real-time application, the consequences of deadline miss of a high priority transaction due to priority inversion can be somehow acceptable for soft RTTs, less severe for firm RTTs, and catastrophic for hard RTTs. Deadline miss of soft RTT results into the degraded value as a function of time. Deadline miss of firm RTT leads to zero value while that of hard RTT leads to a negative value. Zero value means that the firm RTTs are not at all useful in case of their deadline misses. A negative value means that adverse effect of hard RTT's deadline miss can never be compensated; and therefore, algorithms need to be developed so that such transactions must be completed before their deadlines irrespective of any possible priority inversion. Therefore, the enhanced set of transaction scheduling algorithms should be designed to deal with the priority inversion based on the type of RTTs.

All applications access databases only by invoking the transaction. Transactions are basically responsible for the setting of the value of real-world objects. This paper surveys the work on transaction scheduling protocols in time-constrained databases with a focus on solutions to the priority inversion problem. Despite its importance, the priority inversion aspect is not given adequate attention in most of the real-time protocols/algorithms appeared till date. From historical point of view, readers may refer [1,26–43] for knowledge about all aspects of databases.

Rest of the survey paper discusses existing transaction scheduling protocols with pin-pointed focus on priority inversion problem. It (survey) is organized as follows. Section 2 illustrates the fundamentals of transaction processing in time-constrained databases. This is followed by a discussion of the priority inversion problem due to execute–execute conflict while scheduling concurrently executing transactions in Section 3. In Section 4, the problem of priority inversion is discussed in the context of execute–commit conflicts. Later, in Section 5, the real-time optimistic concurrency control techniques for resolving priority inversion problem are discussed. Furthermore, in Section 6, pin-pointed research directions are suggested for future considerations in the context of advancing transaction scheduling algorithms. Finally, in Section 7, our findings have been concluded.

2. Fundamentals of transaction processing in time constrained databases

The general DRTDBS framework involves multiple concrete physical databases consisting of numerous sites situated at multiple locations and connected through a network. So, the database – a collection of data items – is partitioned across multiple sites.

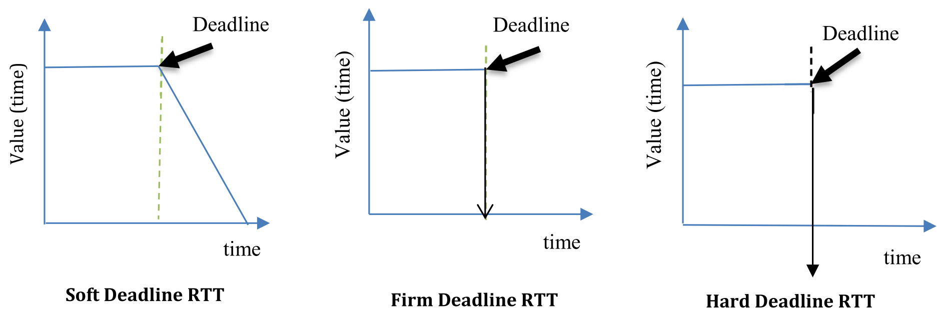

Fig. 1. Common transaction deadline representation with value function.

A sub-collection of data items physically residing at an individual site forms a local database for that site. So, one can say that the whole logical database is fragmented and placed at multiple locations based on the users need and size of relation. Directory, at each site, contains all such information regarding fragmentation of the database.

Fig. 2 represents the interaction among major DRTDBS components involving the execution of distributed RTTs. As shown in Fig. 2, a site may commonly include a resource manager (RM), a transaction manager (TM), a message server and a scheduler [44, 45]. Each site may run several transactions concurrently. Execution of any transaction must transform the underlying database(s) atomically from one state of consistency to another new state of consistency.

The TM is accountable for originating transactions with the transaction identifiers and labeling priorities to them. Each generated transaction is integrated with a deadline. Execution of a transaction follows Cohort-Coordinator based distributed transaction execution framework. A site at which distributed transaction (consisting of the set of local and/or remote data operations) is submitted for execution, must have one coordinator and one cohort. The coordinator first spawns a set of cohorts depending on prerequisite data item list and then coordinates their execution; it performs no database operation (read and/or write). The cohorts are initiated and executed at the sites where the prerequisite data physically reside. Therefore, for any distributed transaction, there must be one cohort and coordinator residing at the originating site (parent site) and at least one cohort at another site. The cohorts are activated by the TM at the participating sites to perform their part of operations. The access list of data is all the data required to be used during the execution of the transaction.

Two categories of transactions have been considered by researchers: local and/or global one. Each individual local transaction is basically a single cohort and a coordinator (processes) executing at its submission site also known as the parent site. The cohort of local transaction requires to access the local data item(s) only; thus, does not need even a single message to be communicated for its execution. On the other hand, each global transaction (distributed transaction) is composed of a set of cohorts to be executed at numerous sites. So, the global transaction is made up of a single coordinator and multiple cohorts; every participating cohort executes at different sites in comparison to their sibling cohorts including one at the parent site. The global transaction requires to access both the local as well as remote data items; thus, needs an exchange of a finite number of messages in between data sites (consisting of cohorts and a coordinator) for the execution of the global transaction.

A message server is associated with each site which works as an interface to connect with all other sites in the system through message passing technique (sending and receiving of messages). In general, the network is presumed to be reliable which means that each message without being lost is disseminated within a finite duration of time with no network partition.

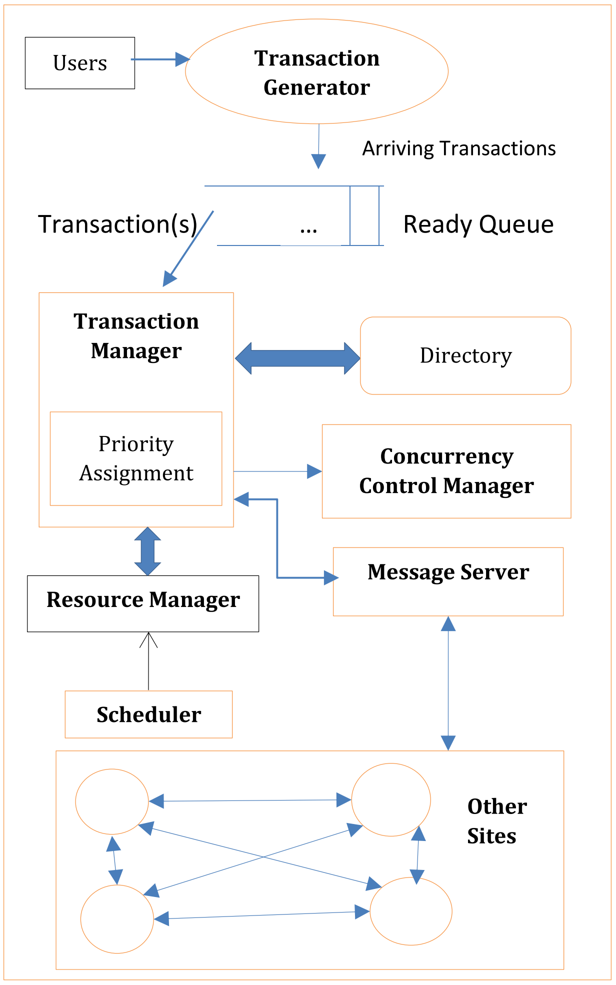

Fig. 2. The DRTDBS model for execution of distributed transactions.

Each cohort performs database operations on data items of their interest. At a site, concurrent data accesses requested by cohorts are regulated by a scheduler. This resource scheduler allows access to the data item(s) as per the concurrency control algorithm. Many concurrency control algorithms were developed to fulfill ever-growing needs with the increasingly complex nature of databases. At first, such algorithms were developed for conventional (non-real time and/or centralized and/or single-site) DBS and then for advanced applications. The concurrency control algorithms ensure isolation during concurrent execution of transactions by controlling the interaction among them using a serializable schedule; and thus maintain database consistency [20,46, 47]. Isolation, which is an integral part of database transactional

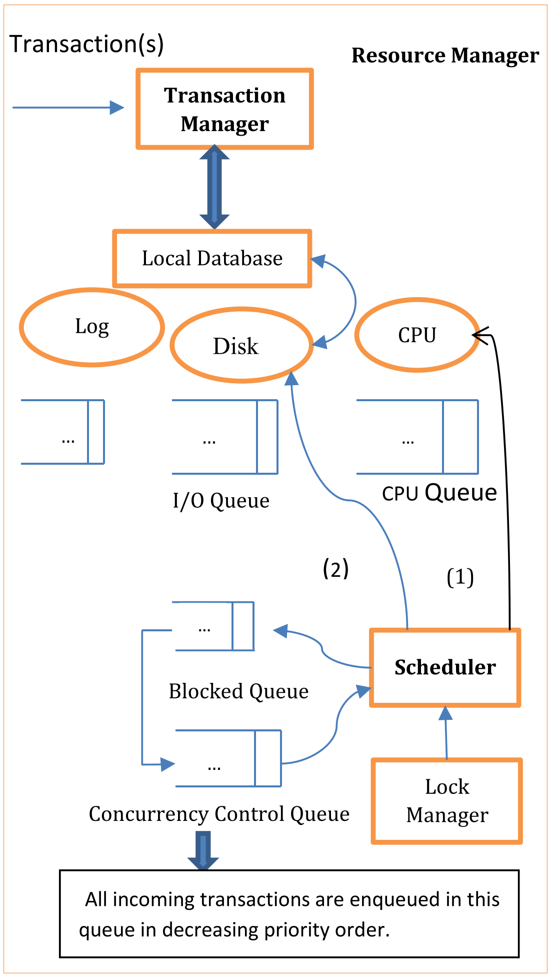

Fig. 3. Detailed working of resource manager.

properties, ensures that the concurrently executing transactions must not interfere among one another — specifically in terms of accessing intermediate data of one executing transaction by other executing transaction [48]. It defines that when the changes made by a database operation (Read/Write) on some data item become visible to others. As per the ANSI/ISO SQL standard, the isolation levels are enlisted and given as Serializable – the highest level of isolation, Repeatable reads, Read committed and Read uncommitted – the lowest level of isolation.

The functioning of RM is diagrammatically represented in Fig. 3. Each site's RM provides services of the I/O for modifying data and services of the CPU for processing data. The scheduling of CPU and I/O has been performed using the queues available at the CPU and I\O respectively. The cohorts are ordered in these queues based on their priorities. The CPU is scheduled using preemptive high priority scheme. This scheme says that any high priority cohort requiring the CPU, on entering in the concurrency control queue, immediately preempt the low priority cohort which is utilizing the CPU. However, if a transaction from the concurrency control queue requiring access to CPU has low priority value then it is blocked from accessing the CPU. The blocked cohort may resume its normal execution only when its priority level becomes highest among all the cohorts in the concurrency control queue or if the CPU queue becomes empty. In simple terms, the CPU is allocated to cohorts on the basis of their priorities. The communication message is always provided higher priority over the other requests of processing.

Transactions in the centralized RTDBS always execute in a sequential manner. Nonetheless, the execution of any distributed transaction may be implemented using one of the two types of DRTDBS models: sequential or parallel. It is quite obvious that the execution of local transactions takes place in an identical manner for both the execution models; the only difference that lies is in the execution of global transactions by these execution models. In the case of the sequential model, all the associated cohorts of a distributed transaction execute in order, and only a single cohort remains active at one point of time. Moreover, the coordinator may start the commit procedure only after the execution of the last cohort in the sequence. In the parallel model of transaction execution, coordinator of a global transaction spawns all the cohorts together at once with the help of sending messages to remote data sites, i.e. cohorts also participating in the completion of a distributed transaction, to make them all active at the same time. It (coordinator) also lists each and every single operation required to be executed at that data sites and then cohorts at respective data sites may possibly start their execution in a parallel fashion. The assumption in case of the parallel model is that the cohort is not required to communicate with its sibling cohorts; operations going to be performed during the execution phase of one cohort are not dependent on the outcome of the operations done by the other sibling cohorts located at geographically distant data sites. At a site, any cohort is said to be completed whenever all of its operations are performed. A completed cohort may then inform the coordinator by sending the 'WORKDONE' message. The coordinator can initiate the 2- Phase Commit (2PC) protocol [49] only after receipt of the 'WORKDONE' messages from all its participating cohorts.

The natural question that may arise here is which one amongst these two models is most suitable for DRTDBS environment. Initially, the comparative study of these two models is done in the context of concurrency control protocols and it has been claimed that the performance of DRTDBS remains similar for both the models [47,50]. The performance improvement of the pessimistic concurrency control (PCC) protocols is marginally better for the parallel execution model, in general than that of the sequential execution model [51]. This improvement in result with the parallel execution model is mainly because of fewer conflicts intuitively due to shorter lifetimes of transactions. Later, it has been confirmed that parallel DRTDBS model is considerably better in comparison to serial DRTDBS model in the context of the overall transaction processing [52,53]. In the parallel transaction execution model, the cohorts execute in parallel and consequently lessen the response-time of the distributed transaction with which they are associated. The commit processing time required is almost equal for both the models [54]. In sequential transaction execution, the benefits are seen mainly due to the active abort policy. In parallel transaction execution, however, the active abort policy does not improve the performance. This is because even if a cohort informs the coordinator about the abort, the chances are that the coordinator has already sent the PREPARE message to all the sibling cohorts ahead of receiving such a message. Therefore, in the parallel case, there are no benefits due to the active abort policy, and hence the performance of real-time commit is less impressive as compared to the sequential case.

Numerous concurrency control algorithms were designed to overcome the database inconsistency problem by ensuring the serializability amongst concurrently executing transactions [31, 55]. These schemes have relied upon a lock-based setting. The non-real time DBS mostly uses 2 Phase Locking (2PL) [56] to ensure isolation correctness criteria among concurrently executing transactions. The 2PL employs 'blocking' to resolve a conflict among transactions. The 2PL and its variants are prevalent conventional lock-based concurrency control algorithms. However, they become unsuitable for the RTDBS environment where a transaction is coupled with a priority. The transaction already holding the data item can have lower priority as compared to the other transaction(s) requesting it. Thus, the execution of transactions with higher priorities can be blocked by a transaction with lower priority. It is extremely detrimental in an environment of real-time and widely recognized as a priority inversion problem.

3. Execute–execute conflict with pessimistic concurrency control protocols

The conventional lock-based concurrency control algorithms, particularly the 2PL and its variants, suits well for conventional databases. The Static 2PL (S2PL) and Dynamic 2PL (D2PL) are the two main variants of the 2PL with different characteristics. They mainly differ in the context of how locks are granted to transactions. In the S2PL, all the data items required by transactions are presumed to be held prior to the start of their executions [57]. In the D2PL, transactions acquire locks dynamically on request and release locks upon abort or commit. The D2PL protocol not only suffers from local deadlock but also incurs large message overhead in enabling locking of data items. However, the S2PL is free from local deadlock and requires comparatively lesser time & message overhead for setting locks. This prompted the research community to look at the S2PL as a more suitable variant to go with as compared to the D2PL when it comes to the real-time environment where transactions are associated with deadlines. Though the ~~D2PL~~ has the above positive features, on utilizing it for RTDBS and DRTDBS, it may lead to the prolonged blocking of high priority transaction due to its either-all-locks-granted-or-none behavior which does not allow it to lock any of the requested data items even when only a single data item is not available at the time of request.

The real-time concurrency control algorithms' resolve transactional data conflicts through priority-based decision makings. The data conflict does not always lead to the problem of priority inversion. But this problem mainly occurs due to data conflict. It is completely acceptable if some high priority transaction is holding a data item, and low priority transaction is blocked after requesting access to it at some later stage. But the opposite case is extremely undesirable where low priority transaction gets preference over high priority transaction for whatsoever reasons. The execute–execute conflict is a data access conflict that may occur among concurrently executing transactions [58–60]. This conflict may lead to the priority inversion problem. In past, noteworthy research works have been carried out by many researchers to resolve this conflict [61–66].

In the RTDBS, to eliminate the priority inversion, in the perspective of the execute–execute conflict, the 2PL with High Priority (2PL-HP) protocol is proposed which decides that the requesting cohort will be either blocked or permitted to lock the data item (s) [67,68]. The requesting cohort is blocked if a data item requested by it is already held by some other executing cohort with higher priority. However, to ensure the uninterrupted execution of the requesting cohort, the lock-holding cohort may get aborted if its priority is lower than the priority of the lock requesting cohort. Further, in case of multiple read requests followed by multiple write requests on the same data item, a new incoming cohort intending to read can join the group of lock-holding reader cohorts only if its priority is higher than that of all the cohort waiting for the write lock [63,69]. The 2PL-HP is free from the ill effects of priority inversion resulting due to execute–execute conflict. However, it may cause wastage of system resources due to unnecessary abort [68].

The 2PL Wait Promote (2PL-WP) [67,68] lock-based protocol is based on the priority inheritance mechanism [70]. The 2PL and the 2PL-WP are similar in the sense that they both resolve the data conflicts (if any) occurring among transactions through blocking of a requesting transaction irrespective of their priorities. Though, the 2PL-WP is a priority sensitive protocol since it considers the priorities of conflicting transactions in resolving the priority inversion problem if occurred. It reduces the duration of priority inversion of the requesting transaction with a high priority which has been blocked by some transactions with low priority [66,71]. J. Huang et al. [66,71] studied the priority inversion resulting due to execute–execute conflict among concurrently executing transactions in a centralized soft RTDBS environment. Two basic policies i.e. priority inheritance and priority abort policy were examined for controlling priority inversion. When priority inheritance is employed, the priority of the low priority lock holding transaction is inherited to that of the high priority lock requesting transaction(s) in case of any such data conflict; and thereby priority inversion duration is reduced. When Priority-Abort is employed, low priority lock holding transaction is immediately aborted in case of data conflict with some high priority transaction, and therefore, this policy is free from priority inversion. The working principle of both the concurrency control algorithms (Priority-Abort and 2PL-HP [67]) is the same; only their names are different. Further, the conditional priority inheritance (CPI) policy has been proposed that leverages the best of each of the above two basic policies i.e. reducing resource wastage (Priority-Inheritance) and respecting priority (Priority-Abort). It is claimed that the CPI policy improves the system performance over Priority-Inheritance and Priority-Abort protocols.

O. Ulusoy et al. [59,60] also investigated the problem of priority inversion resulted due to execute–execute conflict between concurrently executing transactions in RTDBS environment and proposed a locking protocol by using the concept of Data-Priority (DP). The DP protocol is free from deadlock (since each transaction has unique priority and transaction associated with lower priority never blocks transaction with higher priority). It prevents the problem of priority inversion through scheduling lock requests for data items based on their priorities. Implementation of the above locking protocol was done by maintaining a transaction list for each distinct data item. This list resides at the site where the data item physically exists. The list contains identity and priority of all the transactions that are supposed to lock (or else have locked) the data item. All the transactions belonging to the transaction list are sorted on the basis of their priority values. A transaction with the highest priority from the transaction list decides the priority of data item. Every transaction in the system has a data item access-list required during its execution. Transaction scheduler updates the list during initialization and commit of the transaction. Although it is shown that the DP protocol performs slightly well in comparison to Always Block (AB) protocol, Priority Inheritance (PI) protocol and Priority Based (PB) locking protocol, it creates an extra overhead of keeping the updated transaction list for each individual data item that is acquired (or requested) by any of the active transactions in the system. Practically, in any system, number of data items is always considerably larger than the number of active transactions. Here, the PB protocol's working is the same as that of 2PL-HP, and Priority-Abort; only they are differently named by researchers, and the active transaction is an executing transaction.

The Real-Time S2PL (RT-S2PL) protocol [72], unlike the 2PL-HP, utilizes non-preemptive method to resolve the case of conflict among transactions — lock obtained by any transaction with low priority cannot be preempted as a consequence of data conflict with high priority transaction(s). In the RT-S2PL, likewise the S2PL, all data items required by any ready-to-execute transaction must be locked ahead of the start of its execution. Here, a priority value is associated with each data item which is same as that of the priority of the transaction having highest priority from the group of transactions that have requested for a lock on the respective data item. During the execution of requesting transaction, if conflict occurs due to any of its requested data items – data items are either currently held by some low priority transaction, or if some of its requested data items have a priority value higher than that of the requesting transaction – then none of the requested data items is locked by a requesting transaction. However, if any of the data items required to be locked by the blocked requesting transaction has priority value lower than the priority of requesting transaction, the priority value of such data item will be inherited so that its priority may become the same as the priority of requesting transaction. By doing this, in contrary to the S2PL, the blocking time of transaction having high priority is lessened since the data item required by it cannot be locked by a transaction which has a priority lower than its own priority at an intermediate stage. Moreover, the RT-S2PL uses a sequential locking method in order to avoid distributed deadlock situation in DRTDBS environment. A centralized version of the RT-S2PL, named Centralized RT-S2PL (CRT-S2PL), is also developed for RTDBS environment. All such features of the RT-S2PL and the CRT-S2PL protocols designate them as appropriate choices for DRTDBS and RTDBS environments respectively. The brief comparison of the above discussed lock-based concurrency control protocols is presented in Table 1.

To control the priority inversion, many priority ceiling protocols [73–75] have been proposed, in the RTS. The priority ceiling value is defined for each data item, and it is equal to the priority of the highest priority task requiring a lock on the data item. The two decisive features of these protocols are single blocking and freedom from deadlock. However, these protocols cannot directly be integrated with RTDBS as they cannot guarantee the serializable executions of transactions. A Reduced Ceiling Protocol (RCP) has been proposed that guaranteed the schedulability aspects of hard RTTs by means of reserving the data items for them as well as by eliminating the blocking time from soft RTTs [76]. It is also claimed that the RCP can give guarantee for the better performance of the hard RTTs, and in unison reduces the soft RTTs miss percentage. Here, a mixture of both the hard and soft RTTs is considered for the study. An affected set priority ceiling (ASPC) protocol has been presented in [77]. The ASPC is compatible with semantic concurrency control protocols supporting data logical consistency. It prevents deadlock and bounds priority inversion. In the series of the ASPC protocol, a Distributed Affected Set Priority Ceiling (DASPC) protocol was presented for a new class of real-time object-oriented (RTOO) systems [78]. The ASPC was specifically proposed for the single-site system while DASPC for a distributed system. Priority ceiling technique is combined with semantic concurrency control in both the protocols for avoiding unbounded priority inversion and preventing deadlock as well. The semantic concurrency control is implemented to gain benefits of added concurrency.

L. Sha et al. have extended the original priority ceiling protocol for hard RTDBS and proposed Read/Write Priority Ceiling Protocol [16]. Despite equipped with above desirable features, it may not be widely accepted since its inefficient utilization of CPU during input–output operations due to its conservative nature of limiting effective concurrency to one. The operations performed on any complex RTDBS may be categorized as database and/or non-database operations. Only a few research works have been done in the context of the handling of non-database operations; though, research in this direction may be useful to improve underlying system performance. The Non-Database Operations Aware Priority Ceiling Protocol (N-DOAP) is a recent effort in this direction [79]. In the N-DOAP protocol, the database operations are given preference over non-database operations. Consideration of the aspect of non-database operations resulted in the improvement of performance.

To conclude Section 3, we can say that the priority inversion resulting due to execute–execute conflict is widely discussed in the literature in the past. As discussed in this section, researchers have extensively focused on controlling the priority inversion problem through applying different approaches — mainly immediate abort policy as in 2PL-HP, priority inheritance policy as in 2PL-WP, and set of priority ceiling protocols. Moreover, it could be said that the priority ceiling approach may not be suitable to solve priority inversion problem in time-constrained databases due to limited concurrency provided by them. Some performance improvements were seen when the immediate abort policy and priority inheritance policy is wisely integrated. Therefore, some more work in this direction may result in improved system performance.

4. Execute–commit conflict with pessimistic concurrency control protocols

The overhead of concurrency control in multi-site DRTDBS is much higher than in single-site RTDBS as the data required by a transaction may reside at different sites. Here, a more complex type of conflict i.e. execute–commit conflict may occur where the committing low priority lock holding cohort may block executing high priority lock requesting cohort(s). Although the priority inversion problem (in the context of the execute–execute conflict) can be eliminated using the 2PL-HP, this problem cannot be completely eliminated in case of the execute–commit conflict using the 2PL-HP. Resolution of priority inversion because of the execute–commit conflict is more difficult/complex than that of execute–execute conflict. In DRTDBS with the 2PL-HP, the cost of restarting lower priority prepared transaction is much higher as it may have a number of participants at different sites which lead to greater system complexity. Additional sources of unpredictability may result from distributed deadlock, processing and management of distributed transactions and availability of data items. The management of inherited priorities and the blocking status of the transactions in a distributed environment require additional communications.

The 2PL-HP has widely been studied in RTDBS [19] but only a few works have been done to extend its applicability in a distributed environment [31]. Such extension requires consideration of how and when one shall restart a global transaction with sub-transactions running at different sites [57]. The complication lies on two ends. First, for the conflict resolution, transaction restarts occur much more often than in a non-real-time system. Second, it is hard to evaluate whether a restart is beneficial or not. For example, a global transaction consisting of multiple cohorts requiring lots of system resources to complete. Ironically, it is due to its size which makes the global lengthy transaction a likely easy target of data conflict and abort. Restarting a longer transaction annuls all the works done on it and is thus very wasteful. Worse still, it may not be possible to restart a global transaction that has partially committed (E.g., when it is executing two-phase commit and that some of its cohorts have already been locally committed [80].). Therefore, restarting a committing transaction could be disastrous and unwise. Since the delay caused by completing the committing transaction is mild, the system might as well let it finish. The 2PL-HP also suffers from the problems of deadlock and the cyclic restart. These problems are major reasons behind wastage of system resources, and subsequently deadline miss of transactions.

To overcome execute–commit conflicts, the lender–borrower approach is commonly used in next-generation commit protocols

Table 1
A summary of concurrency control protocols for handling execute–execute conflicts.

Protocol	Lock based concurrency control protocols and handling of data conflicts					
	Suitable target environment	Priority sensitive	Priority inversion	Local deadlock	Preemption	Resource wastage
D2PL	Conventional database	N/A	N/A	√	N/A	Low
S2PL	RTDBS	No	Yes	×	×	Medium
RT-S2PL	RTDBS	Yes	Yes	×	×	Medium
2PL-HP	RTDBS	Yes	No	√	√	High
2PL-WP	RTDBS	Yes	Yes	√	×	Low
CPI	RTDBS	Yes	Yes	√	√	Medium

√ - Protocol exhibit the characteristic, × - Protocol does not exhibit the characteristic, N/A — Characteristic is not applicable for protocol.

— which are not only of optimistic nature but also intended to satisfy real-time constraints. Such real-time commit protocols permit prepared cohorts to lend their uncommitted data items to executing cohorts, obviously in case of execute–commit conflicts. This reduces data inaccessibility. However, they usually block the borrower from sending a PREPARED message to its respective coordinator until the completion of the lender transaction and thus increases the borrowing transaction's commit time. Overall, they improve the performance of the system.

To incorporate the lender–borrower approach with distributed real-time commit protocols, the 2PL-HP is also modified accordingly. As a result, an Extended 2PL high priority (E2PL-HP) protocol has been proposed in [52,81,82] with no concrete comparison with 2PL-HP. The E2PL-HP has following salient features.

1. After receiving PREPARE message from the coordinator, cohort releases all its locks of shared mode but retains its exclusive locks until it implements the final global decision received from the coordinator.
2. A cohort already in the prepared state cannot be forcefully aborted irrespective of any priority inversion at its end.
3. Requesters can lock the uncommitted data item(s), which is/are already been modified by a prepared cohort(s) using the Lender-Borrower Approach [52,53,83] in a controlled manner.

However, E2PL-HP suffers from the problem of database inconsistency. This makes it an inappropriate choice for the distributed real-time environment. To understand the above problem, suppose a read-only cohort has released its lock from data item 'O' on the receipt of PREPARE message from the coordinator. Then after, some other cohort (T_{ic}), belonging to a global transaction T_i, locked the data item O in exclusive mode, modified it and committed. There is a possibility that the read-only cohort may not complete its execution even after the completion of T_i, whose cohort (T_{ic}) has updated the above-uncommitted data item 'O'. As a result, the database enters into an inconsistent state. Though the 2PL-HP is free from priority inversion when used in RTDBS, its successor the E2PL-HP is not free from priority inversion when used in DRTDBS.

The static 2PL with priority inheritance (S2PL-PI) protocol [48] is proposed for DRTDBS. It specifically minimizes the resource wastage by avoiding unnecessary abort of transactions through the use of priority inheritance policy in an optimal way. The S2PL-PI also overcomes the starvation problem associated with lengthy transactions only to some extent. This protocol is developed for parallel DRTDBS framework. Through avoiding the deadlock with the help of controlled locking and reducing the negative effect of starvation with fair resource allocation strategy, the Controlled Avoidance of deadlock and starvation causing Resourceful Conflict resolution between Transactions (CART) protocol [84] minimizes the miss percentage of transactions. It is basically achieved by reducing resource wastage. A sequential distributed transaction execution model has been considered for assessment of the performance of the CART protocol and comparison has been made with other previous protocols.

Ramesh Gupta et al. first proposed a real-time variant of the classical 2PC protocol, named Optimistic (OPT) protocol [85]; this protocol is specifically designed to meet the requirements of firm-deadline-based DRTDBS by reducing the negative effect of inherent priority inversion problem. The OPT protocol provides features like controlled optimistic access to uncommitted data, active abort, and silent kill. This protocol allows a high priority cohort to borrow/access the uncommitted data items held by some prepared low priority cohort. Such lending by a prepared cohort creates a dependency between transactions. It means the fate of borrower transaction depends on the final outcomes of transactions from which it had borrowed the data items — if final outcomes of all the lender transactions are 'commit', then only the borrower transaction can start its commit processing. Thus, OPT makes the borrower blocked until the time lenders complete their execution; this optimism is advantageous only when lenders successfully complete their executions. While the policy of using uncommitted data items may result into the chain of dependencies which may sometimes cause cascading aborts, the OPT protocol restricts the length of chain to one by not allowing the borrowers to simultaneously become a lender. Therefore, it does not suffer from cascading abort problem. It provides a significant performance improvement over 2PC.

Moreover, two variants of OPT is suggested; Healthy-OPT and Shadow-OPT. In Healthy-OPT, every executing transaction is assigned a health factor (HF), and a transaction can become lender only when its HF value is greater than or equal to some threshold value. Obviously, the performance of the system heavily varies with change in the chosen threshold value. In Shadow-OPT, the cohort creates a shadow (replica of itself) at the instance it borrows some data item. Moreover, the original version of the cohort continues its execution as usual even after borrowing uncommitted data items while the shadow cohort is blocked right after it is forked off as a separate cohort. In case of lenders successfully commit, the shadow cohort is discarded since optimistic borrowing has done its job well. Otherwise, if any of the lenders (who has a dependency with this cohort) aborts, the shadow cohort is activated, and the original cohort gets aborted. Thus, Shadow-OPT saves the cohort from restarting and provides a feature so that the cohort can resume the processing from the point the borrowing is done in case of an unsuccessful lending–borrowing event. In the end, detailed simulation results showed that Healthy-OPT performs better than OPT and Shadow-OPT.

The "Permits Reading of Modified Prepared Data for Timeliness" protocol was abbreviated as PROMPT protocol [52]. It is exclusively an extension/integration of the work done in [54, 85–90] discussing OPT protocol and its variants. The PROMPT protocol includes features like optimistic access to uncommitted data, active abort, silent kill, and healthy lending. Though, the PROMPT protocol remains the standard for a long time to assess the performance of other real-time commit protocols that came after it, several shortcomings of this protocol are raised by researchers with time. Some key developments are discussed below.

To reduce data inaccessibility and priority inversion duration, the Double Space Commit (2SC) protocol [81], an optimistic real-time commit protocol specifically designed for the high performance distributed RTTs, demonstrates two properties. At first, unlike PROMPT, it permits a non-healthy prepared transaction to lend its 'modified prepared data' to the transactions in its commit dependency set (CDS). At second, in case of the abort of the prepared transaction, only transactions in its abort dependency set (ADS) are aborted while transactions in its CDS execute as usual. The 2SC protocol performs better than PROMPT under all load conditions. The 'Static two-phase locking and high priority-based, Write-update type, Ideal for Fast and Timeliness' commit protocol has been abbreviated as SWIFT protocol [53]. It first analyzed all type of dependencies arising during execute–commit conflicts. Here, the cohort's execution phase is divided in locking phase and processing phase. Also, a WORKSTARTED message is sent just before starting of the processing phase of the cohort in place of a WORKDONE message. This, in turn, improves system performance by overlapping the cohort's WORKSTARTED message transmission time with its processing time. The SWIFT protocol performance has been compared with PROMPT and 2SC protocol and simulation results show up to 10% performance improvement in transaction miss percentage. Extended SWIFT protocol [91] proposed by Ankit Aakash et al. discussed the database inconsistency problem resulting due to Read-Read access in the DRTDBS. This protocol uses the Lender-Borrower approach to deal with the priority inversion.

ACTIVE protocol [92] categorizes borrowers as commit dependent or abort dependent. The commit dependent borrower is further permitted to lend data; thereby, reducing the inaccessibility of data. In this work, a borrowing factor is computed for each borrower. On the basis of it, the lock on requested data item can be granted to second-step borrowers fruitfully. In PERDURABLE protocol [93], a lender cohort already lent data to abort dependent borrower cohort can lend the same data in read mode to multiple borrowers. The availability of data has also been improved by allowing already locked data items (not utilized during the processing phase) to be freed well before starting of the commit phase. Then after, an Improved Data Lending based Distributed Real-Time Commit (IDRC) protocol [83] has been developed; it lessens the effect of data conflicts on the DRTDBS performance by improving the data accessibility in comparison with similar earlier protocols. It does so by permitting the borrower cohort of commit dependent type (already lent the data item to the abort dependent borrower cohort) to lend the same data item in multiread mode to second-step borrower participant cohorts.

The Shadow-PROMPT protocol is based on the concept of shadowing. It forks off shadow cohort every time any cohort borrows a data item irrespective of the type of dependency imposed on it due to the lending–borrowing event. The Dependency Sensitive Shadow-SWIFT (DSS-SWIFT) [94] protocol improves the Shadow-PROMPT protocol by considerably reducing the overhead of shadow creation and management through forking off cohorts only when it is required. It allows the cohort to fork of its shadow only if dependency is of 'abort type'. No shadow cohort is forked off in case of commit dependency since here an abort of the lender cohort does not result in the abort of the borrower cohort. As claimed, the DSS-SWIFT performed well as compared to Shadow-PROMPT. Though DSS-SWIFT solved the problem of unnecessary shadow creation with Shadow-PROMPT to some extent, it still creates non-fruitful shadows in some cases. This problem is further resolved in "Shadow, Piggy bag, Elemental External Dependency Inversion and in Time Yielding" (SPEEDITY) protocol [95], which limits the task of replica creation to the point where abort dependent borrower cohort will fork of its shadow only if its deadline is greater than a specific value i.e. ($T_{shadow\_creation\_time}$). In SPEEDITY, the cost of achieving the objective of creating only fruitful shadows is an increased dependency management overhead with the introduction of two new dependencies — Begin-on-Abort Dependency (BAD) and Commit-on-Termination Dependency (CTD). Marginal performance improvement is seen with SPEEDITY protocol over DSS-SWIFT.

In [52], the Priority Inheritance Commit (PIC) protocol is also been discussed as an alternative approach to increase data accessibility and reduces the negative effects of priority inversion. It does so by allowing all the cohorts and a coordinator of conflicting low priority transaction to execute at the highest priority (highest priority is a priority value which is highest amongst the priority values of all the high priority cohorts blocked by conflicting prepared low priority cohort). After completion of low priority transaction, its inherited priority value reverts to original priority value, and data item(s) locked by it get released. Thus, the high priority transaction blocked by some low priority transaction will get the required data items somewhat earlier. It has been said that the performance results with PIC protocol are equivalent to that of with 2PC protocol.

As an extended version of the PIC protocol, a 1-Phase PIC (OPPIC) protocol [70] is proposed. The OPPIC protocol is, in principle, opposite to the PIC protocol in terms of the dissemination of PRIORITY-INHERIT message between participants of the distributed transaction, at least one of whose cohort encountered a priority inversion problem. In this case of occurrence of priority inversion, the conflicting low priority cohort sends the PRIORITY-INHERIT message parallelly to all its other sibling cohorts as well as to coordinator. It is beneficial in dealing with priority inversion as it notably minimizes the overall completion time of the distributed firm RTT that has suffered from execute–commit conflict. This protocol may suffer from many other problems as well. Therefore, to make the alternative approach of priority inheritance logically feasible, there is a wide range of issues requiring researchers' attention. They can be listed as follows.

i. The possibility of multi-cohort single transaction priority inversions may result in a drastic message overhead due to the requirement of implementing multiple priority inheritance events at a single transaction.
ii. The abrupt increase in the number of priority inheritance events may result into the increase in competition to access named data items which in turn negatively affects the execution of transactions that naturally had higher priorities [66,71].
iii. The priority de-boosting process involved in this scheme may also result in an overhead.
iv. There is a chance of occurrence of the deadlock due to holding data item by a cohort at one site while waiting for data item by a cohort at another site where all such cohorts are meant for completion of a single transaction (distributed hold-and-wait).
v. There is a chance of starvation with the transaction requesting write access in the case where multi-read accesses are followed by this conflicting write access request.
vi. Even though the priority inheritance approach may lead to restricted priority inversion, the time span of priority inversion is highly dependent on two things — a number of transactions executing concurrently and number of locks held by them.

From the above studies, it could be inferred that several factors especially cyclic restart, deadlock, priority inversion, starvation of lengthy transactions, and chained blocking profoundly influence pessimistic concurrency control protocols' performance. Additionally, the occurrences of execute–commit conflict make the design of concurrency control and commit protocol extensively difficult. The priority inversion problem resulted due to such conflicts raised many unresolved issues in designing both the protocols (concurrency control and commit). A new set of real-time commit protocols were proposed addressing the above issue through either lender–borrower approach or priority inheritance approach. Though a very little research effort is made towards designing priority inheritance-based real-time commit protocols. We believe that any such effort may be a logical extension to the existing literature. To have clear research perspective, Table 2 summarizes all the protocols discussed in Section 4.

5. Data conflict with optimistic concurrency control protocols

The concurrency control schemes can be categorized as follows: (i). PCC scheme, and (ii). OCC scheme. Although almost all commercial applications based on database use PCC schemes, some studies [68] claim that OCC schemes can give better performance. In the PCC scheme, conflict is identified before accessing data item; while in the case of the OCC scheme, conflict is identified after accessing the data item. The OCC with Broadcast Commit (OCC-BC) [96,97] is one of the common variations of the OCC scheme. The OCC-BC protocol divides transaction execution into 3 phases: read phase, validation phase and finally write phase. Here, all the modifications are performed in a transaction's private workspace to ensure consistency of the underlying database. During the read phase, all the operations associated with a transaction are processed in the order. After read phase, the transaction passes into the validation phase, where the data conflicts with other concurrently executing uncommitted transactions are checked. In case a conflict is detected, transactions involved in a conflict with the validating transaction get restarted. At the end, during the write phase, updates made to the transaction's private workspace are reflected in the real database. Such a scheme guarantees that every transaction that enters in the validation phase will eventually commit.

The major advantage of OCC-BC over 2PL-HP is that all possible restarts that may happen during the validation phase will be useful since the system has more information for resolving data conflicts. In simple words, if a validating transaction restarts even a single concurrently executing uncommitted transactions then it is guaranteed that the validating transaction must be committed in future; thus, there is no cyclic restart problem with OCC-BC. However, as a result, OCC-BC leads to a large number of restarts as compared to the 2PL-HP (because of late detection of data conflict and very restrictive validation phase rule). It also does not consider real-time constraints (i.e., priorities) associated with transactions, and therefore, may not be suitable for RTDBS environment [80]. For instance, low-priority transactions (during its validation phase) can unilaterally commit and cause conflicting higher priority transactions to be restarted. It means low priority transaction has undesirably been preferred over high priority transaction(s), and thus priority inversion occurred. Such action may lead to the deadline miss of high priority transactions. A large number of restarts in a real-time environment means increased miss ratio, low resource utilization, and high overhead. So, the problems with OCC-BC are as follow.

1. Inefficient implementation of the validation phase in the absence of a global system view on transactions' read and write sets, and
2. Inherent priority inversion problem as a low priority validating transaction may restart a concurrently executing high priority transaction(s) in case of conflict which is extremely undesirable in a real-time environment.

SACRIFICE, Priority-Abort, and WAIT-50 protocols [68] are developed to overcome the above priority inversion problem associated with OCC-BC. They are specifically based on the validating low priority transaction that has a conflict with a number of high priority transactions. In SACRIFICE, on the occurrence of conflict with any high priority transaction, the validating transaction is restarted immediately. With Priority Abort, the validating transaction is restarted only when all the conflicting transactions have high priority. WAIT-50 leverages the best of both the worlds (SACRIFICE and Priority-Abort) i.e. respecting priority (SACRIFICE) and minimizing number of restarts (Priority-Abort). In WAIT-50, the validating transaction enters in the wait state when number of conflicting transactions with higher priority is greater than or equal to the number of conflicting transactions with lower priority. The WAIT-50 protocol uses a prioritized wait control technique and is claimed to outperform other priority conscious OCC algorithms. However, it does not unilaterally outperform OCC-BC which is a priority unconscious OCC protocol. Moreover, the WAIT-50 protocol may not be suitable and/or efficient in the distributed environment since it requires continuous monitoring of the conflicting transactions. Such a requirement makes the WAIT-50 inefficient where communication among sites is expensive. Here, every data access must be reported to all the validating waiting transactions.

Further, as a part of an important development in RTDBS, the OCC with Timestamp Intervals (OCC-TI) protocol is proposed [98]. As the name suggests, it resolves the conflicts among transactions using timestamp intervals. Unlike OCC-BC, the OCC-TI protocol does not require the serialization order of transactions to be same as validation order [99]. Moreover, the OCC-TI is not a pure optimistic protocol since it performs some of the conflict checks in the first phase (read phase) itself. This aids in avoiding unnecessary execution of the non-serializable transaction to its validation phase; thus, OCC-TI avoids resource wastage. The OCC-TI is claimed to perform better than OCC-BC and WAIT-50 [100]. Unlike WAIT -50, the OCC-TI and OCC-BC, both, do not use priority information for resolving conflicts amongst concurrently executing transactions and guarantee the commitment of validating transactions. While the OCC-BC restarts all the transactions that conflict with a validating transaction, the OCC-TI restarts the irreconcilably conflicting transactions only.

The OCC — Adaptive Priority (OCC-APR) is an extension of the OCC-TI protocol [100]. Unlike OCC-TI, OCC-APR is a priority sensitive protocol which uses the priority information in making a decision about whether a validating transaction can be restarted or not. The validating transaction will be restarted if at least one of the transactions in the irreconcilably conflicting set has priority value higher than the validating transactions, and the validating transaction is very likely to commit after the restart. Though OCC-APR is an obvious and intelligent real-time variant of OCC-TI, it does not lead to any such performance improvement over OCC-TI.

The OCC with Dynamic Adjustment (OCC-DA) protocol is proposed to reduce number of restarted transactions with the help of adjustment of the serialization order dynamically. The OCC-DA employs the dynamic timestamp assignment policy. Moreover, there is no need of adjustment of a timestamp in case of data conflicts while being in the read mode. The outperformance of OCC-DA over OCC-TI can be seen with the help of results obtained during their performance studies [101].

The OCC-DA and OCC-TI both use dynamic adjustment. However, unnecessary adjustments are made by them in case of aborts of the validating transaction. Unnecessary adjustments are made by OCC-TI even in the read phase also. Such unnecessary adjustments may later responsible for unnecessary restarts. To resolve the problem of unnecessary restarts, the OCC-DATI (Optimistic CONCURRENCY CONTROL with Dynamic Adjustment of

Table 2
Priority inversion in the perspective of the execute–commit conflict — A broader perspective.

Role of existing protocols for the resolution of priority inversion in the perspective of the execute–commit conflict		
Role of concurrency control protocol		
Pessimistic approach	2PL-HP [31,80]	**Advantage:** Eliminates priority inversion problem that has occurred as a result of the execute–execute conflict.
		Disadvantages: 1. Priority inversion may still occur due to execute–commit conflict. 2. Suffer from problems such as unnecessary abort, deadlock, cyclic restart, and starvation
	E2PL-HP [52]	The adverse effect of priority inversion with 2PL-HP is reduced by using the Lender–Borrower Approach.
	S2PL-PI [48]	1. Avoids unnecessary abort 2. Overcomes starvation problem to some extent.
	CART [84]	1. In contrary to the above 3 protocols, CART is developed serial DRTDBS model. 2. Reduce resource wastage 3. Minimized transaction miss percent
Role of commit protocol		
Lender-borrower approach	OPT [85] PROMPT [52] 2SC [82]	Haritsa et al. at first used a lender–borrower approach in transaction scheduling and proposed OPT and PROMPT protocols to overcome execute–commit conflict. However, lending was very restrictive.
	SWIFT [53] ACTIVE [92] PERDURABLE [93] IDRC [83]	These protocols are enhancements of the PROMPT protocol. The only purpose of all these protocols was to optimize/increase lending while ensuring consistency.
	Shadow-PROMPT [52] DSS-SWIFT [94] SPEEDITY [95]	These set of protocols specifically utilize the concept of shadowing. Moreover, SPEEDITY is the most improved version of Shadow-PROMPT till date.
Priority inheritance approach	PIC [52]	Requires two phases to disseminate priority inheritance information. This drastically increases the overall transaction execution time in any such case and eventually defeats the purpose behind it.
	OPPIC [70]	Requires only one phase to disseminate priority inheritance information. Thus, OPPIC propose a novel enhancement in the PIC protocol.

the Serialization order using Timestamp Intervals) protocol has been proposed which uses a deferred dynamic adjustment of serialization order among the conflicting transactions [102]. The OCC-DATI uses dynamic adjustment of the serialization order in the same fashion as done in OCC-TI [103]. In OCC-DATI, all the timestamp interval adjustments are performed to temporal variables. The all conflicting active transactions' timestamp interval are adjusted when the validating transaction is guaranteed to commit. Unlike OCC-TI, a conflict check is performed only at the validation phase. No conflict check is performed during read phase. All of these developments affect the performance of RTDBS and result in the reduced transaction miss percent. The OCC-DATI is different from OCC-DA [101] in many aspects. For the implementation of dynamic adjustment of the order of the serialization, the time intervals as used in OCC-DA are adopted instead of a dynamic timestamp assignment.

Later, an OCC with Real-Time Serializability and Dynamic Adjustment of Serialization Order (OCC-RTDATI) protocol [104] has been proposed that uses real-time serializability for conflict resolution. The validation process is the same as in the case of OCC-DATI and OCC-RTDATI. The OCC-RTDATI uses forward validation technique [105] and is based on the previous OCC-DATI protocol [102]. It differs in the way of resolution of occurred conflict. The OCC-RTDATI utilizes real-time serializability as well as transaction object attributes for conflict resolution. Although OCC has been claimed to outperform locking protocols based on lock allocation in RTDBS, optimistic methods have their own problems of needlessly as well as heavy restarts overhead. The DOCC-DATI (Distributed OCC-DATI) protocol is also proposed based on OCC-DATI protocol. Some new features are added to OCC-DATI protocol to attain distributed serializability and to resolve the problems of the distributed OCC techniques. For clear understanding and detailed discussion of series of developments before OCC-RTDATI, J. Lindstrom's papers [102,104,106–111] provide detailed information.

From the above studies, it could be inferred that especially the number of restarts and useless restarts influence OCC protocols' performance. The OCC schemes received less attention as compared to pessimistic concurrency control algorithms. Till date, OCC scheme and its variants are only studied by the academic and research community; almost all the commercial applications use pessimistic concurrency control algorithms. However, it seems that OCC schemes may be a good choice for time-constrained databases perhaps by wisely integrating them with pessimistic concurrency control algorithms. A summarized report of the aforementioned OCC schemes is presented in Table 3.

6. Scope for future research

Considering an unpredictable system scenario, future time-constrained databases may be operative in adaptable run-time systems that would be capable of handling an unknown and varying number of concurrently executing transactions effectively and efficiently with no inconsistency and priority inversion. Thus, the promising and challenging aims for future research work are listed below.

1. ***Priority inversion:*** The data conflicts (execute–execute & execute–commit conflict) and possible priority inversion because of them are one of the main reasons for degraded system performance. Therefore, existing transaction scheduling algorithms need a fresh look from the perspective of efficiently resolving data conflicts.

Table 3
A summary of OCC protocols handling data conflicts.

Optimistic real-time CC protocols and handling of data conflicts					
Protocol	Priority sensitive	Priority inversion	Restart overhead	Unnecessary restarts	Unnecessary adjustments
OCC	No	Yes	High	√	N/A
OCC-BC	No	Yes	High	√	N/A
OCC-TI	No	Yes	Medium	√	√
OCC-DA	No	Yes	Medium	√	√
OCC-DATI	No	Yes	Low	×	×
SACRIFICE	Yes	No	High	×	N/A
Priority abort	Yes	Yes	Low	×	N/A
WAIT 50	Yes	Yes	Medium	×	N/A
OCC-APR	Yes	Yes	High	×	√
OCC-RTDATI	Yes	Yes	Medium	×	×

√ - Protocol exhibit the characteristic, × - Protocol does not exhibit the characteristic, N/A – Characteristic is not applicable for protocol.

2. There is a need of further investigation to design novel real-time recovery and logging algorithms while considering the priority inversion aspect because the traditional recovery and logging algorithms are not suitable for priority sensitive databases [19,112–114].
3. For time-constrained database applications, the applicability of priority ceiling protocols in controlling priority inversion seems to do well as it can ensure that a transaction with high priority may have to wait for the completion of at most one transaction [115]. But, it badly affects the concurrency level of underlying time-constrained database applications; and therefore priority ceiling approach may not be a good choice in comparison to the priority inheritance approach for controlling priority inversions.
4. The integration of the RTDBS with active databases results in the introduction of the Real-Time Active Database System (RTADBS) [116–120]. Such integration is not free from worries and may lead to even more complex scheduling problems [121,122]. In RTADBS, triggered transactions are a major reason of unpredictability (a sudden increase in the workload on the system and a sudden increase in the probability of data conflicts) [123]. This can affect the transactions deadline miss percentage. Therefore, the role of priority inversion problem can be critical in decreasing the triggered transactions deadline miss percentage by solving the issue of unpredictability resulted due to triggering. Extension of the RTADBS in a distributed environment and consequential problems are also studied by few researchers [124–126].
5. Though minimizing the transaction deadline miss percentage is widely accepted metric for performance evaluation of time-constrained databases, it may be more appropriate to decide priority based on 'deadline' and 'value' of the transactions both. Therefore, the primary objective could be to maximize the value created by transactions that commit before their deadlines [127].
6. Although a few research papers appeared addressing the real-time CPU scheduling, most of them are based on [128]. Till date, no concrete research efforts have been made to develop a specialized real-time CPU scheduling schemes with a focus on priority inversion problem and its effect on the time-constrained distributed database environment.
7. Although several methods have been proposed to integrate the two scheduling algorithms i.e. real-time scheduling and database concurrency control, very few research works have been done to assess the dynamic cost involved in scheduling a transaction [129]. Therefore, research efforts in this direction may be very fruitful.
8. The predictability and consistency both are important factors in the processing of transactions in the real-time environment but sometimes ensuring these two factors require database actions in a conflicting way [21,130,131]. Inconsistency resulting due to multiple accesses of read type on the same data items is to be taken care of wisely and carefully [55]. Proper management of delays and handling of execution time uncertainties connected with scheduling schemes are to be kept an eye on.
9. Today, with the emergence of a set of distributed real-time applications, existing priority assignment policies need to be upgraded to meet the ever-challenging demands [80, 132]. The priority assignment policies should be designed after critically analyzing whether task processing requirements are known in advance or not [133]. While considering growing complexity, especially resulting due to priority inversions, and overload conditions, new priority assignment policies need to be designed allowing extended concurrency among competing transactions.
10. The Petri net model can be established as a standard to 'analytically' evaluation of the performance of transaction processing systems. Petri nets are viewed as promising tools for analysis and modeling of the performance of complex information processing systems with discrete events. These events are characterized as being non-deterministic, parallel, concurrent and/or distributed. [134]. Therefore, they are suitable for describing and analyzing the concurrency control in DDBS [135,136]. Petri Net model has been utilized for the 2-phase commit protocol performance studies for the distributed transaction management in [137]. The colored petri net (CPN) model of distributed transaction based on X/Open standards TX and XA is presented [138].

Considering the wide-spreading impact of database system based applications in recent days, researchers also started working to improve the performance of conventional databases systems by improving classic 2PC and 2PL [139–143]. Furthermore, it will be interesting to widen the discussion of priority inversion problem to energy aware RTDBS query processing [144–149], mobile DRTDBS [150–154], replicated DRTDBS [155,156], nested DRTDBS [157–159], active DRTDBS [126], and mobile ad-hoc network (MANET) databases [160–162]. More promising research efforts are required to develop advanced value-based real-time commercial products.

At last, we can say that researchers are explosively studying the concept of 'blockchain transaction processing' these days [163]. Blockchain was first proposed in 2008 by S. Nakamoto [164]; later, it was implemented in 2009. In recent years, a few noble survey papers appeared discussing the blockchain — security issues, challenges, and opportunities associated with it [165, 166]. The blockchain technology is increasingly becoming a key for wide range of real-life applications.

7. Conclusions

We explored various approaches while considering the impact of prevention or minimization of priority inversion on the system performance especially time-constrained databases (RTDBS and DRTDBS). This survey may be considered as a reference guide in designing both the general as well as application-specific transaction scheduling protocols with a focus on avoiding priority inversion. In the future, we would have to devise mechanisms for avoiding priority inversion completely, perhaps ideally. This could assist in the formulation of implementation guidelines — when priority inversion handling techniques should be used and when they should not be used. While designing priority sensitive transaction scheduling protocols, researchers would have to focus not only on the avoidance of unbounded priority inversion but also on minimization of the priority inversion duration. Addressing such undefined, demanding and conflicting needs will never be an easy task. Additionally, there are many other research problems requiring proper attention for resolving issues related to transaction scheduling.

To conclude, this survey summarizes major research accomplishments and identifies key issues and open research challenges for future research.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported by the Council of Scientific and Industrial Research India [grant number 1061461137].

References

[1] U. Shanker, M. Misra, A.K. Sarje, Distributed real time database systems: Background and literature review, Int. J. Distrib. Parallel Databases 23 (02) (2008) 127–149, Springer Verlag.
[2] R. Elmasri, Fundamentals of Database Systems, Pearson Education India, 2008.
[3] P. Bernstein, E. Newcomer, Principles of Transaction Processing, Morgan Kaufmann, 2009.
[4] R. Ramakrishnan, J. Gehrke, Database Management Systems, McGraw Hill, 2000.
[5] J. Gray, A. Reuter, Transaction Processing: Concepts and Techniques, Elsevier, 1992.
[6] M. Özsu, P. Valduriez, Principles of Distributed Database Systems, Springer Science & Business Media, 2011.
[7] O. Babaoglu, K. Marzullo, F. Schneider, Priority Inversion and Its Prevention, TR 90-1088, 1990.
[8] Ö. Babaoğlu, K. Marzullo, F.B. Schneider, A formalization of priority inversion, Real-Time Syst. 05 (04) (1993) 285–303, 5, 285-303.
[9] B.W. Lampson, D.D. Redell, Experience with processes and monitors in Mesa, Commun. ACM 23 (02) (1980) 105–117.
[10] J.B. Goodenough, L. Sha, The priority ceiling protocol: A method for minimizing the blocking of high priority Ada tasks, ACM SIGAda Ada Lett. VIII (07) (1988) 20–31.
[11] S. Davari, L. Sha, Sources of unbounded priority inversion in real-time systems and a comperative study of possible solutions, ACM Oper. Syst. Rev. (1992) 110–120.
[12] R. Rajkumar, Task synchronization in real-time systems (Ph. D. Dissertation), 1989.
[13] O. Ornvall, Benchmarking Real-Time Operating Systems for Use in Radio Base Station Applications, Department of Computer Science and Engineering, Chalmers University of Technology, Göteborg, Sweden, 2012.
[14] C. Krishna, Real-Time Systems, in: Wiley Encyclopedia of Electrical and Electronics Engineering, 2001.
[15] R. Mall, Real-Time Systems: Theory and Practice, Pearson Education India, 2009.
[16] L. Sha, R. Rajkumar, S.H. Son, C.H. Chang, A real-time locking protocol, IEEE Trans. Comput. 40 (07) (1991) 793–800.
[17] L. DiPippo, V. Wolfe, Real-Time Databases, Database Systems Handbook, Multiscience Press, 1997.
[18] K. Ramamritham, Real-time databases, Distrib. Parallel Databases 01 (02) (1993) 199–226.
[19] K. Ramamritham, S.H. Son, L.C. Dipippo, Real-time databases and data services, Real-Time Syst. 28 (2–3) (2004) 179–215.
[20] S. Pandey, U. Shanker, 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), 2018, pp. 305-309.
[21] Y. Kim, S. Son, Predictability and consistency in real-time database systems, Adv. Real-Time Systems (1995) 509–531.
[22] G. Ozsoyoglu, R.T. Snodgrass, Temporal and real-time databases: A survey, IEEE Trans. Knowl. Data Eng. 07 (04) (1995) 513–532.
[23] U. Shanker, Some performance issues in distributed real time database systems (Ph.D. thesis), Indian Institute of Technology Roorkee, 2008.
[24] S.A. Aldarmi, Scheduling soft-deadline real-time transactions (Ph.D. Thesis), University of York, 1999.
[25] J. Jonsson, Deadline assignment in distributed hard real-time systems with relaxed locality constraints, in: Proceedings - 17th International Conference on Distributed Computing Systems, 1997, pp. 432–440.
[26] C. Mohan, An overview of recent data base research, ACM SIGMIS Database: DATABASE Adv. Inf. Syst. 10 (2) (1978) 3–24.
[27] J. Gray, The transaction concept: Virtues and limitations, in: VLDB, Vol. 81, 1981, pp. 144–154, September.
[28] M. Carey, R. Jauhari, M. Livny, Priority in DBMS Resource Scheduling, University of Wisconsin-Madison, Computer Sciences Department, 1989.
[29] H. Garcia-Molina, B. Lindsay, Research directions for distributed databases, ACM SIGMOD Rec. 19 (4) (1990) 98–103.
[30] C. Mohan, R. Dievendorff, Recent work on distributed commit protocols, and recoverable messaging and queuing, Data Eng. 17 (1) (1994).
[31] B. Kao, H. Garcia-Molina, An overview of real-time database systems, Real Time Comput. 127 (1993) 261–282.
[32] Ö. Ulusoy, Research issues in real-time database systems, Survey paper, Inf. Sci. 87 (1–3) (1995) 123–151.
[33] A. Bestavros, K. Lin, S. Son, Advances in real-time database systems research, in: Real-Time Database Systems, Springer, Boston, MA, 1997, pp. 1–14.
[34] A. Bestavros, K. Lin, S. Son, Real-Time Database Systems: Issues and Applications, Vol. 396, Springer Science & Business Media, 2012.
[35] R. Ginis, V. Wolfe, Issues in designing open distributed real-time databases, in: IEEE Proceedings of the 4th International Workshop on Parallel and Distributed Real-Time Systems, 1996, pp. 106-109.
[36] B. Purimetla, R. Sivasankaran, K. Ramamritham, J. Stankovic, Real-time databases: Issues and applications, Adv. Real-Time Syst. (1995) 487–507.
[37] P. Chrysanthis, G. Samaras, Y. Al-Houmaily, Recovery and performance of atomic commit processing in distributed database systems, Recov. Mech. Database Syst. (1998) 370–416.
[38] J. Stankovic, K. Ramamritham, D. Towsley, Scheduling in real-time transaction systems, Found. Real-Time Comput.: Sched. Resour. Manag. (1991) 157–184.
[39] J. Stankovic, S. Son, J. Hansson, Misconceptions about real-time databases, Computer 32 (6) (1999) 29–36.
[40] J. Haritsa, K. Ramamritham, Real-time database systems in the new millenium, Real-Time Syst. 19 (3) (2000) 205–208.
[41] U. Shanker, M. Misra, A.K. Sarje, Hard Real-Time Distributed Database Systems: Future Directions, IIT Roorkee, India, 2001, pp. 172–177.
[42] S. Pandey, U. Shanker, Transaction execution in distributed real-time database systems, in: Proceedings of the International Conference on Innovations in information Embedded and Communication Systems, 2016, pp. 96-100.
[43] U. Shanker, M. Misra, A. Sarje, Some performance issues in distributed real-time database systems, in: Proc. VLDB Ph.D. Work, Conv. Exhib. Cent. (COEX), Seoul, Korea, 2006.
[44] O.a.B.G. Ulusoy, A simulation model for distributed real-time database systems, in: 25th Annual IEEE Proceedings Simulation Symposium, 1992, pp. 232-240.
[45] W. Haque, P. Stokes, Simulation of a complex distributed real-time database system, Spring Simul. Multiconf.- Soc. Comput. Simul. Int. 2 (2007) 359–366.
[46] P. Bernstein, V. Hadzilacos, N. Goodman, Concurrency control and recovery in database systems, 1987.
[47] K. Lam, C.L. Pang, S. Son, J. Cao, Resolving executing-committing conflicts in distributed real-time database systems, Comput. J. 42 (08) (1999) 674–692.
[48] S. Pandey, U. Shanker, On using priority inheritance based distributed static two phase locking protocol, in: Proceedings of the International Conference on Data and Information System (ICDIS), 2017, pp. 179-188..
[49] Y. Al-Houmaily, G. Samaras, Two-phase commit, in: Encyclopedia of Database Systems, Springer, Boston, MA, 2009, pp. 3204–3209.

[50] Ö. Ulusoy, Processing real-time transactions in a replicated database system, Distrib. Parallel Databases 2 (4) (1994) 405–436.
[51] O. Ulusoy, G. Belford, Real-time lock-based concurrency control in distributed database systems, in: Proceedings of the 12th IEEE International Conference on Distributed Computing Systems, 1992, pp. 136-143.
[52] J.R. Haritsa, K. Ramamritham, R. Gupta, The PROMPT real-time commit protocol, IEEE Trans. Parallel Distrib. Syst. 11 (02) (2000) 160–181.
[53] U. Shanker, M. Misra, A.K. Sarje, SWIFT - A New real time commit protocol, Distrib. Parallel Databases 20 (01) (2006) 29–56.
[54] R. Gupta, Commit processing in distributed on-line and real-time transaction processing systems Sc.(Engg.) Thesis, SERC, Indian Institute of Science, 1997.
[55] P.S. Yu, K.-l. Wu, K.-j. Lin, S.H. Son, On real-time databases : Concurrency control and scheduling, Proc. IEEE 82 (01) (1994) 140–157.
[56] J.M. Faleiro, D.J. Abadi, Rethinking serializable multiversion concurrency control, Vldb 08 (11) (2015) 1190–1201.
[57] K.Y. Lam, Concurrency control in distributed real time database systems (Ph.D. thesis), 1994.
[58] J. Lindström, Using priorities in concurrency control for RTDBS, in: Seminar on Real-Time and Embedded Systems, Department of Computer Science, University of Helsinki, Autumn, 1999.
[59] Ö. Ulusoy, G.G. Belford, Real-time transaction scheduling in database systems, Inf. Syst. 18 (08) (1993) 559–580.
[60] O. Ulusoy, G.G. Belford, Concurrency control in real-time database systems, in: ACM Annual Conference on Communications, 1992.
[61] Y. Tay, N. Goodman, R. Suri, Locking performance in centralized databases, ACM Trans. Database Syst. 10 (4) (1985) 415–462.
[62] J. Haritsa, M. Carey, M. Livny, Dynamic real-time optimistic concurrency control, in: IEEE Proceedings of the 11th Real-Time Systems Symposium, December 1990, pp. 94-103.
[63] J. Haritsa, M. Carey, M. Livny, On being optimistic about real-time constraints, in: Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, 1990, pp. 331-343.
[64] A. Burger, V. Kumar, M.L. Hines, Performance of multiversion and distributed two-phase locking concurrency control mechanisms in distributed databases, Inform. Sci. 96 (1) (1997) 129, 96 (1) 129-152.
[65] S. Chakravarthy, D. Hong, T. Johnson, Real-time transaction scheduling: A framework for synthesizing static and dynamic factors, Real-Time Syst. 14 (2) (1998) 135–170.
[66] J. Huang, J.A. Stankovic, K. Ramamritham, D. Towsley, On using priority inheritance in real-time databases, in: Real-Time Systems Symposium, 1991, pp. 210–221.
[67] R.K. Abbott, H.G. Molina, Scheduling real-time transactions: a performance evaluation, ACM Trans. Database Syst. 17 (03) (1992) 513–560.
[68] J.R. Haritsa, M.J. Carey, M. Livny, Data access scheduling in firm real-time database systems, Real-Time Syst. 04 (03) (1992) 203–241.
[69] W. Abid, M. Mhiri, M. Salem, E. Bouazizi, F. Gargouri, A feedback control scheduling architecture for real-time ontology, in: 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE), IEEE, 2017, pp. 1–7.
[70] S. Pandey, U. Shanker, 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, 2018.
[71] J. Huang, J.A. Stankovic, K. Ramamritham, D. Towsley, B. Purimetla, Priority inheritance in soft real-time databases, Real-Time Syst. 04 (03) (1992) 243–278.
[72] K.-Y. Lam, S.-L. Hung, S.H. Son, On using real-time static locking protocols for distributed real-time databases, Real-Time Syst. 13 (02) (1997) 141–166.
[73] T. Baker, Stack-based scheduling of real-timeprocesses, Real-Time Syst. 03 (01) (1991) 67–99.
[74] M. Chen, K.J. Lin, Dynamic priority ceilings: A concurrency control protocol for real-timesystems, Real-Time Syst. 02 (04) (1990) 325–346.
[75] L. Sha, R. Rajkumar, J.P. Lehoczky, Priority inheritance protocols: An approach to real-time synchronization, IEEE Trans. Comput. 39 (9) (1990) 1175–1185.
[76] K. Lam, T. Kuo, W. Tsang, Concurrency control for real time database systems with mixed transactions, in: Real-Time Computing Systems and Applications, 1997, pp. 96–103.
[77] M.A. Squadrito, Extending the priority ceiling protocol using read/write affected sets (Master of Science in Computer Science), University of Rhode, Island, 1996.
[78] M. Squadrito, L. Esibov, L.C. DiPippo, V.F. Wolfe, G. Cooper, B. Thurasingham, M. Milligan, The affected set priority ceiling protocols for real-time object-oriented concurrency control, Int. J. Comput. Syst. Sci. Eng. 14 (04) (1999) 227–239.
[79] R.K. Singh, S. Pandey, U. Shanker, 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, 2019, (in press).
[80] V. Lee, K. Lam, B. Kao, Priority scheduling of transactions in distributed real-time databases, Real-Time Syst. 16 (1) (1999) 31–62.
[81] B. Qin, Y. Liu, High performance distributed real-time commit protocol, J. Syst. Softw. 68 (02) (2003) 145–152.
[82] B. Qin, Y. Liu, J. Yang, A commit strategy for distributed real-time transaction, J. Comput. Sci. Tech. 18 (5) (2003) 626–631.
[83] S. Pandey, U. Shanker, IDRC: A distributed real-time commit protocol, Procedia Comput. Sci. 125 (2018) 290–296.
[84] S. Pandey, U. Shanker, CART: A real-time concurrency control protocol, in: Bipin C. Desai, Jun Hong, Richard McClatchey (Eds.), 22nd International Database Engineering & Applications Symposium (IDEAS 2018), ACM, New York, NY, USA, 2018.
[85] R. Gupta, J. Haritsa, K. Ramamritham, Revisiting commit processing in distributed database systems, ACM SIGMOD Rec. 26 (2) (1997) 486–497.
[86] R. Gupta, J. Haritsa, Commit processing in distributed real-time database systems, in: Proc. of National Conf. on Software for Real-Time Systems, Cochin, India, 1996.
[87] R. Gupta, J. Haritsa, K. Ramamritham, S. Seshadri, Commit processing in distributed real-time database systems, in: Real-Time Systems Symposium, 1996, pp. 220–229.
[88] R. Gupta, J. Haritsa, K. Ramamritham, More Optimism About Real-Time Distributed Commit Processing, rtss IEEE, 1997.
[89] R. Gupta, J. Haritsa, K. Ramamritham, S. Seshadri, Commit Processing in distributed real-time database systems, in: Technical Report TR -1996-01, Database System Lab, Supercomputer Research Centre, IISc Banglore, 1996.
[90] J. Haritsa, K. Ramamritham, R. Gupta, Characterization and optimization of commit processing performance in distributed database systems, in: Technical Report, University of Massachusetts, 1998.
[91] A. Aakash, A. Gaurav, A. Gupta, B. Kumar, Extended SWIFT: A real time commit protocol, SCIT J. XV (2015).
[92] U. Shanker, N. Agarwal, S. Tiwari, P. Goel, P. Srivastava, ACTIVE-A real time commit protocol, Wirel. Sensor Netw. 2 (3) (2010).
[93] U. Shanker, B. Vidyareddi, A. Shukla, Perdurable: A real time commit protocol, Recent Trends Inf. Reuse Integr. (2012) 1–17.
[94] U. Shanker, M. Misra, A. Sarje, R. Shisondia, Dependency sensitive shadow SWIFT, in: Database Engineering and Applications Symposium IDEAS'06. 10th International IEEE, 2006, pp. 273–276.
[95] S. Agrawal, U. Shanker, A. Singh, A. Anand, SPEEDITY-A real time commit protocol, Int. J. Comput. Appl. (0975–8887) 1 (3) (2010) 86–93.
[96] D.A. Menascé, T. Nakanishi, Optimistic versus pessimistic concurrency control mechanisms in database management systems, Inf. Syst. 7 (1) (1982) 13–27.
[97] J.T. Robinson, Design of Concurrency Controls for Transaction Processing Systems, No. CMU-CS-82-114, Carnegie-Mellon Univ Pittsburgh Pa Dept of Computer Science, 1982.
[98] J. Lee, S. Son, Using dynamic adjustment of serialization order for real-time database systems, in: Real-Time Systems Symposium. Proceedings, IEEE, 1993, pp. 66–75.
[99] J. Lee, S. Son, Concurrency control algorithms for real-time database systems (Doctoral dissertation), University of Virginia, 1994.
[100] A. Datta, S.H. Son, V. Kumar, Is a bird in the hand worth more than two in the bush? limitations of priority cognizance in conflict resolution for firm real-time database systems, IEEE Trans. Comput. 49 (5) (2000) 482–502.
[101] K.-W. Lam, K.-Y. Lam, S. Hung, An efficient real-time optimistic concurrency control protocol, in: Proceedings of the Active and Real-Time Database Systems (ARTDB-95), Springer, London, 1996, pp. 209–225.
[102] J. Lindstrom, K. Raatikainen, Dynamic adjustment of serialization order using time-stamp intervals in real-time databases, in: Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications. RTCSA'99, IEEE Computer Society Press, 1999, pp. 13–20.
[103] J. Lee, S. Son, Performance of concurrency control algorithms for real-time database systems, in: V. Kumar (Ed.), Performance of Concurrency Control Mechanisms in Centralized Database Systems, Prentice-Hall, 1996, pp. 429–460.
[104] J. Lindström, K. Raatikainen, Using real-time serializability and optimistic concurrency control in firm real-time databases, in: Proceedings of the 4th IEEE International Baltic Workshop on DB and IS BalticDB & IS, 2000, pp. 1-5.
[105] T. Härder, Observations on optimistic concurrency control schemes, Inf. Syst. 9 (2) (1984) 111–120.
[106] J. Lindstrom, Extensions to optimistic concurrency control with time intervals, in: Proceedings of the 7th International Conference OnReal-Time Computing Systems and Applications, IEEE Computer Society Press, 2000, pp. 108–115.

[107] J. Lindstrom, Integrated and adaptive optimistic concurrency control method for real-time databases, in: Proceedings of the 8th International Conference on Real-Time Computing Systems and Applications, 2002, pp. 143-151.
[108] J. Lindström, T. Niklander, Benchmark for real-time database systems for telecommunications, in: International Workshop on Databases in Telecommunications, in: Lecture Notes in Computer Science, vol. 2209, Springer, Berlin, Heidelberg, 2001, pp. 88–101.
[109] J. Lindström, T. Niklander, P. Porkka, K. Raatikainen, A distributed real-time main-memory database for telecommunication, in: International Workshop on Databases in Telecommunications, Vol. 1819, Springer, Berlin, Heidelberg, 1999, pp. 158–173.
[110] J. Lindstrom, K. Raatikainen, Using importance of transactions and optimistic concurrency control in firm real-time databases, in: Proceedingsof the 7th International Conference on Real-Time Computing Systems and Applications, 2000, pp. 463-467.
[111] J. Lindström, Optimistic concurrency control methods for real-time database systems (Ph.D. thesis), Department of Computer Science, University of Helsinki, 2003.
[112] R. Sivasankaran, K. Ramamritham, J. Stankovic, Logging and recovery algorithm for real-time database, in: Tecnical Report, Department of Computer Science. University of Massachusetts, 1997.
[113] R. Sivasankaran, K. Ramamritham, J. Stankovic, System failure and recovery, in: Real-Time Database Systems, Springer, Boston, MA, 2002, pp. 109–124.
[114] L. Shu, J. Stankovic, S. Son, Achieving bounded and predictable recovery using real-time logging, Comput. J. 47 (3) (2004) 373–394.
[115] T. Kuo, Y. Kao, C. Kuo, Two-version based concurrency control and recovery in real-time client/server databases, IEEE Trans. Comput. (2003) 506–524.
[116] J. Eriksson, Real-time and active databases: A survey, in: Active, Real-Time, and Temporal Database Systems, Springer, Berlin, Heidelberg, 1998, pp. 1–23.
[117] R. Sivasankaran, B. Purimetla, J. Stankovic, K. Ramamritham, D. Towsley, Design of RADEx: Real-Time Active Database Experimental System, University of Massachusetts, 1994.
[118] A. Datta, S. Mukherjee, I. Viguier, Buffer management in real-time active database systems, J. Syst. Softw. 42 (3) (1998) 227–246.
[119] A. Datta, S. Son, A study of concurrency control in real-time, active database systems, IEEE Trans. Knowl. Data Eng. 14 (3) (2002) 465–484.
[120] Y. Qiao, K. Zhong, H. Wang, X. Li, Developing event-condition-action rules in real-time active database, in: Proceedings of the 2007 ACM symposium on Applied computing, pp. 511-516.
[121] K. Lam, T. Lee, S. Son, READS: a prototyping environment for real-time active applications, in: Dings of the Eighth International Workshop on Database and Expert Systems Applications, IEEE, 1997, pp. 265–270.
[122] K. Ramamritham, R. Sivasankaran, J. Stankovic, D. Towsley, M. Xiong, Integrating temporal, real-time, an active databases, ACM Sigmod Rec. 25 (1) (1996) 8–12.
[123] K.Y. Lam, T.S. Lee, Approaches for scheduling of triggered transactions in real-time active database systems, in: 24th Proceedings of Euromicro Conference, IEEE, 1998, pp. 476–483.
[124] B. Purimetla, R.M. Sivasankaran, J.A. Stankovic, K. Ramamritham, D. Towsley, A study of distributed real-time active database applications, in: IEEE Workshop on Parallel and Distributed Real-Time Systems, Vol. 75, 1993.
[125] Ö. Ulusoy, Transaction processing in distributed active real-time database systems, J. Syst. Softw. 42 (3) (1998) 247–262.
[126] K. Lam, G. Law, V. Lee, Priority and deadline assignment to triggered transactions in distributed real-time active databases, J. Syst. Softw. 51 (1) (2000) 49–60.
[127] J. Haritsa, M. Carey, M. Livny, Value-based scheduling in real-time database systems., The VLDB J.—Int. J. Very Large Data Bases 2 (2) (1993) 117–152.
[128] H. Tokuda, T. Nakajima, P. Rao, Real-time mach: Towards a predictable real-time system, in: USENIX Mach Symposium, 1990, pp. 73–82.
[129] D. Hong, T. Johnson, S. Chakravarthy, Real-time transaction scheduling: a cost conscious approach, in: Proceedings of the ACM SIGMOD international conference on Management of data, Vol. 22(2), 1993, pp. 197-206.
[130] Y.K. Kim, Predictability and consistency in real-time transaction processing (Doctoral dissertation), University of Virginia, 1995.
[131] S.H. Son, Y.K. Kim, Predictability and consistency in real-time database systems, in: Proceedings of InfoScience, 1993, pp. 225-232.
[132] U. Shanker, M. Misra, A.K. Sarje, Priority assignment heuristic to cohorts executing in parallel, in: 9th International Conference on World Scientific and Engineering Academy and Society (WSEAS), 2005.
[133] J. Haritsa, M. Livny, M. Carey, Earliest deadline scheduling for real-time database systems, in: Proceedings Twelfth IEEE Real-Time Systems Symposium, 1991, pp. 232-242.
[134] O. Diallo, J. Rodrigues, M. Sene, Performances evaluation and Petri nets, Model. Simul. Comput. Netw. Syst. (2015) 313–355.
[135] D. Haryono, Petri Net modelling of concurrency control in distributed database system, J. Sist. Komput. 2 (2) (2012) 35–42.
[136] Y. Han, C. Jiang, X. Luo, A study of concurrency control in Web-based distributed real-time database system using extended time Petri nets, in: Parallel Architectures, Algorithms and Networks. IEEE Proceedings. 7th International Symposium, 2004, pp. 67-72.
[137] B. Sarkar, N. Chaki, Transaction management for distributed database using petri nets, Int. J. Comput. Inf. Syst. Ind. Manag. Appl. (IJCISIM) 2 (69) (2010) 69–76.
[138] M. Iwaniak, W. Khadzhynov, Colored Petri net model of x/open distributed transaction processing environment with single application program, in: International Conference: Beyond Databases, Architectures and Structures, Springer, 2014, pp. 20–29.
[139] S. Gupta, M. Sadoghi, Easycommit: A non-blocking two-phase commit protocol, in: International Conference on Extending Database Technology (EDBT), 2018, pp. 157–168.
[140] S. Gupta, M. Sadoghi, Efficient and non-blocking agreement protocols, Distrib. Parallel Databases (2019) 1–47.
[141] M. Sadoghi, S. Blanas, Transaction processing on modern hardware, Synth. Lect. on Data Manag. 14 (2) (2019) 1–138.
[142] A.K. Pandey, S. Pandey, U. Shanker, 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, 2019 (in press).
[143] R. Harding, D.V. Aken, A. Pavlo, M. Stonebraker, An evaluation of distributed concurrency control, Vldb 10 (05) (2016) 553–564.
[144] K. Kang, Enhancing timeliness and saving power in real-time databases, Real-Time Syst. 54 (2) (2018) 484–513.
[145] K. Kang, Reducing deadline misses and power consumption in real-time databases, in: IEEE Real-Time Systems Symposium (RTSS), 2016, pp. 257–268.
[146] W. Kang, J. Chung, Energy-efficient response time management for embedded databases, Real-Time Syst. 53 (2) (2017) 228–253.
[147] W. Kang, S. Son, Power-and time-aware buffer cache management for real-time embedded databases, J. Syst. Archit. 58 (6–7) (2012) 233–246.
[148] V. Legout, M. Jan, L. Pautet, Scheduling algorithms to reduce the static energy consumption of real-time systems, Real-Time Syst. 51 (2) (2015) 153–191.
[149] M. Völp, M. Hähnel, A. Lackorzynski, Has energy surpassed timeliness? scheduling energy-constrained mixed-criticality systems, in: 20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014, pp. 275–284.
[150] K. Lam, T. Kuo, W. Tsang, G. Law, Transaction shipping approach for mobile distributed real-time databases, in: International Conference on Database and Expert Systems Applications, Springer, Berlin, Heidelberg, 1999, pp. 932–941.
[151] K. Lam, T. Kuo, G. Law, W. Tsang, A similarity-based protocol for concurrency control in mobile distributed real-time database systems, in: International Parallel Processing Symposium, Springer, Berlin, Heidelberg, 1999, pp. 329–338.
[152] K. Lam, T. Kuo, W. Tsang, G. Law, Concurrency control in mobile distributed real-time database systems, Inf. Syst. 25 (4) (2000) 261–286.
[153] K. Lam, T. Kuo, Mobile distributed real-time database systems, in: Real-Time Database Systems, Springer, Boston, MA, 2002, pp. 245–258.
[154] X. Lei, Y. Zhao, S. Chen, X. Yuan, Concurrency control in mobile distributed real-time database systems, J. Parallel Distrib. Comput. 69 (10) (2009) 866–876.
[155] M. Xiong, K. Ramamritham, J.R. Haritsa, J.A. Stankovic, MIRROR: A state-conscious concurrency control protocol for replicated real-time databases, Inf. Syst. 27 (04) (2002) 277–297.
[156] Y. Wei, A. Aslinger, S. Son, J. Stankovic, 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), 2004.
[157] M. Abdouli, B. Sadeg, L. Amanton, Scheduling distributed real-time nested transactions, in: Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC), 2005, pp. 208–215.
[158] M. Abdouli, B. Sadeg, L. Amanton, A. Alimi, A system supporting nested transactions in DRTDBSs, in: International Conference on High Performance Computing and Communications, Springer, Berlin, Heidelberg, 2005, pp. 888–897.
[159] S. Moon, S. Lee, A reliable nested transaction model with extension of real-time characteristics, in: Reliable and Autonomous Computational Science, Springer, Basel, 2011, pp. 123–142.
[160] Z. Xing, L. Gruenwald, S. Song, An optimistic concurrency control algorithm for mobile ad-hoc network databases, in: InACM Proceedings of the Fourteenth International Database Engineering & Applications Symposium, 2010, pp. 199-204, August.

[161] Z. Xing, L. Gruenwald, An energy-efficient concurrency control algorithm for mobile ad-hoc network databases, in: International Conference on Database and Expert Systems Applications, Springer, Berlin, Heidelberg, 2011, pp. 496–510.
[162] Z. Xing, L. Gruenwald, Managing concurrent execution of transactions in mobile ad-hoc network database systems: an energy-efficient approach, Distrib. Parallel Databases 31 (2) (2013) 183–230.
[163] S. Gupta, M. Sadoghi, Blockchain transaction processing, in: Encyclopedia of Big Data Technologies, 2019.
[164] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, 2008, https://bitcoin.org/bitcoin.pdf.
[165] Z. Zheng, S. Xie, H. Dai, H. Wang, Blockchain challenges and opportunities: A survey, Int. J. Web Grid Serv. (2016) 1–25.
[166] I. Lin, T. Liao, A survey of blockchain security issues and challenges, IJ Netw. Secur. 19 (5) (2017) 653–659.