

On Using Similarity for Concurrency Control in Real-Time Database Systems

Kam-yiu Lam and Wai-cheong Yau

Department of Computer Science

City University of Hong Kong

83 Tat Chee Avenue, Kowloon

HONG KONG

email : cskylam@cityu.edu.hk

Abstract

Most of the proposed concurrency control protocols for real-time database systems (RTDBS) are based on serializability theorem. Owing to the unique characteristics of real-time database applications and the importance of satisfying the timing constraint of the transactions, serializable concurrency control protocols are not suitable for RTDBS for most cases. In this paper, another notion of correctness, *similarity*, is used for concurrency control in a RTDBS, for instance, a stock trading database system. *Similarity* is a less restrictive notion comparing with serializability. By studying the correctness requirements of the stock trading database applications, a real-time two phase locking protocol, High Priority 2 Phase Locking (H2PL) is re-defined based on *similarity*. Although the new protocol cannot ensure serializability, the concurrency of the system is higher and the amount of inconsistency in the database is tolerable. On the other hand, the performance of the whole system can be much improved.

Keywords : *real-time database systems, applications, concurrency control, Similarity*

1 Introduction

Real-time database systems (RTDBS) are generally defined as the database systems supporting time-constrained transactions. The timing constraints are usually expressed in the form of deadlines. If the deadlines cannot be met, the usefulness of completing the transactions and the correctness of the systems will be greatly affected [Best96, Ozso95, Shu92, Son88, Yu94]. One of the most important issues in the design of RTDBS is transaction scheduling which objectives are to satisfy the timing constraint of the transactions and at the same time to maintain database consistency [Bern87, Gray93].

Database consistency can be achieved with the use of concurrency control protocols [Gray93]. However, the protocols for traditional database systems are not suitable for RTDBS as the priority of transactions are completely ignored in scheduling data to the transactions [Yu94]. If the two phase locking protocol (2PL) [Bern87] is used for RTDBS, priority inversion may occur [Rajk91] which is highly undesirable to real-time scheduling as it can greatly degrade the schedulability of the adopted CPU scheduling algorithm and affect the system performance [Liu73, Stan95].

In recent years, different real-time concurrency control protocols have been proposed [Agra95, Best95, Hari92, Huan92, Lam95, Lee94, Ulus94]. Most of them are based on 2PL owing to its popularity and simplicity in implementation [Eswa76]. The correctness notion adopted in these protocols is serializability [Bern87]. For example, in High Priority Two Phase Locking (H2PL), if the requesting lock is being held by a lower priority transaction, the lower priority transaction will be restarted [Abbo92]. Serializable schedules can still be obtained from H2PL by enforcing the rules for setting and releasing locks in traditional 2PL. Although serializability has been shown to be suitable for traditional database systems, it is too restrictive for RTDBS [Garh94] due to the unique characteristics of real-time database applications as compared with traditional database applications [Kim95, Rama93, Rama95].

To improve the system performance, one recent suggestion is to use *similarity* as the correctness notion for concurrency control in RTDBS [Kuo92, Kuo94]. Although *similarity* is proposed to be a more suitable correctness notion for RTDBS, up to now, very little of work has been done on applying *similarity* in real-time database applications. It is also not clear how *similarity* can be used to improve the performance of real-time database applications. In this paper, we have defined a real-time concurrency control protocol based on *similarity* for a real-time database application, a stock trading database system. Illustrations are given to show how *similarity* can be applied in the application. Simulation experiments have been performed to investigate how *similarity* improves the system performance.

The rest of the paper is organised as follows. Section 2 discusses the problems of concurrency control in RTDBS. Section 3 introduces a real-time database application, a stock trading database system. Section 4 discusses how *similarity* can be applied in the stock trading database system. Section 5 describes the RTDBS model for the stock trading database system using *similarity* and the performance results. Section 6 discusses the limitations of the simulation results and future directions. The conclusions are in Section 7.

2 The Problem

The previous work on the study of RTDBS is concentrated on the design of concurrency control protocols. They assume that the only difference between RTDBS and traditional database systems is on the deadlines of the transactions. Based on this assumption, different real-time concurrency control protocols have been proposed by modifying the concurrency control protocols for traditional database systems such as 2PL [Abbo92, Huan92] and optimistic method [Hari92, Lee94]. Their objectives are to minimise the number deadline missing transactions and maintain database consistency. They resolve the problem of priority inversion [Rajk91] by blocking or restarting transactions. They use serializability as the notion of correctness as it is commonly accepted as the standard of correctness for concurrency control in conventional database systems [Bern87]. Owing to the presence of different uncertainties, i.e., the number of required data objects of a transaction, the arrival pattern of the transactions and their resource requirement, it is very difficult to ensure that all the deadlines of the transactions can be met. Thus, most of the proposed real-time concurrency control protocols are for soft or firm RTDBS [Yu94].

In order to improve the system performance and reduce the number of deadline missing, new correctness notion for concurrency is required for RTDBS. Although maintaining database consistency is important, meeting transaction deadline is also very critical as the usefulness of the transactions will be greatly affected if their deadlines are missed. On the other hand, the problem of having an inconsistent database is less serious in RTDBS [Kim95]. In a real-time database, some data objects, called *temporal data objects*[Kim95, Rama93], are used to monitor the status of some external objects. This relationship with the external environment is called *external consistency* [Lin89] which cannot be maintained by serializable executions of transactions. Instead, it is achieved by timely update of the data objects [Kim95]. Due to the presence of temporal data, the impact of inconsistent database is much smaller than that in conventional database systems as the consistency can be restored simply by the next updates.

One recent suggestion on the correctness for concurrency control in data-intensive systems is *similarity* [Kuo92]. It is a less restrictive notion of correctness. It is suitable for RTDBS as one of the main characteristics of RTDBS is that they have to respond to the changing external environment. The values of many temporal data objects in a real-time system are unable to be updated continuously as the update process itself introduces a delay. So, there already exists a discrepancy in the values of the temporal data objects between the actual values of the objects in the external environment. By assuming that a small discrepancy is tolerable for most applications and data values that are slightly different in age or in precision are often considered to be the same, the concept of *similarity* is defined. By using *similarity* as the correctness notion for concurrency control, it has been claimed that the degree of concurrency can be increased. However, what are the benefits and practical issues in using *similarity* have not been discussed in previous studies.

3 Case Study

In this section, a stock trading database system is introduced. It is a RTDBS [Adel95] as missing transaction deadlines can seriously affect the usefulness of the system. The stock trading database system consists of a number of external monitors, a number of terminals and a central database server. The database is maintained by the central database server. Some of the data objects in the database are used to reflect the current price of the stocks in the stock market. They are temporal data objects. The system subscribes a number of stocks which it is interested in. Whenever there is a change in the price of its subscribed stocks in the stock market, the external monitors will capture the new price of the stock in the stock market and send the update to the central database server. The query terminals generate different types of queries and retrievals. Some of them examine the prices of the stocks for buying or selling of stocks.

3.1 Transaction Classes

In the system, there exists different kinds of transactions with different characteristics. To simplify our study, they are classified into three classes:

Class 1 - Updates and user transactions

Class 2 - Access transactions

Class 3 - Regular retrievals

Class 1 - Updates and Queries

i) Update transactions

Whenever there is a change in the stock price of those stocks subscribed by the system, an update transaction will be generated from the external monitor to update the database. They are higher priorities transactions and have tight deadlines in order to keep the database up-to-date (maintaining the external consistency). Assigning higher priorities to them can ensure that all the updates can be applied in the database on time.

ii) User transactions

User transactions against the stock database will be created on a random ad-hoc basis from the terminals to get information about the stock market such as the price of a particular stock. They have tight deadlines in order to get up-to-date information for trading decision making. Similar to the updates, they are high priority transactions as missing their deadlines renders the information useless and affects decision making.

Class 2 - Access Transactions

They are generated by the activities in the stock trading system. They may be an analysis of the changes in the prices of the stocks or a recalculation of indexes. They usually have to access a number of stocks.

Class 3 - Regular retrievals

Regular retrievals are initiated by the query terminals for regular information retrieval. They are created by financial institutes or stock trading companies.

3.2 The Scheduling Problem

Assuming that there is a stock called Aloca. Its current price is \$60.0. It is assumed that there are five transactions waiting for processing at the current time in the order specified below:

T1: An update with \$60.25

T2: An update with \$60.375

T3: An inquiry for latest price traded

T4: A routine (regular) retrieval

T5: An update with \$60.875

Scenario 1 : Serializable transactions process

It is assumed that 2PL is used for concurrency control to ensure serializability. Following the rules in 2PL, T1 and T2 have to be executed serially as both of them want to update the same data. After their completion, the read transactions, T3 and T4, can be executed concurrently. Then, T5 can

be executed. If a write transaction required 0.1 second to complete, the read transactions (T3 and T4) have to wait for at least 0.2 second. In a stock trading market, this is not desirable as a lot of opportunities will lose within 0.2 second. Also, serial execution of transactions affects the system throughput. The new value from T5 cannot be updated into the database after 0.4 second.

Scenario 2 : Non-serializable transactions process based on Similarity

In the design of non-serializable approach based on *similarity*, the following assumptions are made:

- (1) the regular retrieval (T4) does not care about the freshness of the data objects as it is for reference only and no serious decision will be drawn from it. The information is considered to be correct as long as it is fresh enough.
- (2) the inquiry (T3) is needed for making decision on buying or selling stocks. Although the information is also for reference, it is needed in urgent in order to be useful for decision making.
- (3) the update transactions, T1 and T2, do not have significant difference in the reference point of view as it is assumed that a difference of \$0.2 is tolerable in the market according to the reference information of the stock Aloca. Anyway, after the commitment of the update transaction, T5, the values of \$60.25 and \$60.375 will no longer be useful.

Concurrent execution of T1 and T2 may violate serializability. However, this can reduce the blocking time of transactions and increase their chances to meet their deadlines. The database consistency can be restored after the commitment of T5. Thus, the questions are whether the temporary inconsistency is tolerable and how critical is it for the transactions to meet their deadlines. If the trade off is within the tolerance limit, concurrent execution of transactions is more beneficial.

4 Similarity in Stock Trading Database System

Based on [Kuo92], we define the notion of *similarity* for concurrency control in the stock trading database system. *Similarity* is defined in the context that some values of a data object are considered interchangeable to the system users even though they are not the same. It can be applied to the data objects which store the prices of stocks.

DEFINITION : Two values of a data object are *similar* if all the transactions that may read them consider them the same and will proceed to identical processing.

Example. If all traders will make the same decision when the price of Aloca is \$60.0 or \$60.1, these two values of Aloca are considered to be *similar*.

DEFINITION: Two transactions are *similar* if they are accessing data objects which are *similar*.

Thus, *similarity* is defined not by the value of the data object itself, but by how the value is used. It is an application dependent property. Based on the property, a tolerance limit is defined to determine whether the different values of a data object are *similar* or not. The limit is called *similarity boundary* [Kuo92].

In the context of concurrency control, *similarity* is a relationship between transactions and the view perceived by them. If the inconsistency observed by a transaction is within the *similarity* boundary, the inconsistency will be considered tolerable and concurrent access of data object in incompatible modes [Bern87] will be allowed. For example, if a transaction is updating a data object and another transaction wants to read the same data object, the second transaction has to be blocked according to 2PL. However, if *similarity* is used and the new value of the data object is *similar* to the original value of the data, the second transaction will be allowed to read the original value or the new value from the first transaction. In this way, the concurrency of the system will become higher.

When incorporating *similarity* into a concurrency control protocol, we have to identify under which conditions *similarity* checking should be performed and how to perform the checking. For example, if High Priority 2PL (H2PL) [Abbo88] is used, *similarity* checking are performed whenever there is a lock conflict. If the lock requesting transaction is an updating transaction, the conflicting transactions will be considered to be *similar* if:

- (1) the new value of a data object is *similar* to the original value of the data; and
- (2) the new values of all the updating transactions are *similar*.

If the lock requesting transaction is a read transaction, they will be *similar* if the original value of the data object is *similar* to the new values of all other updating transactions. If the conflicting transactions are not *similar*, the conventional ways for solving conflict in H2PL will be used.

5 Performance Studies

To study the benefit of using *similarity* for concurrency control in the stock trading database system, a model of the system has been developed and simulation experiments have been performed.

5.1 Real-time Database System Model

It is assumed that there are three client sites connected to the central database server in the stock trading database system. In each client site, three classes of transactions (as specified in section 3) are generated by three generators independently. All transactions are sent to the database server for processing.

The inter-arrival time of the transactions from a generator follows exponential distribution. Each transaction is assigned a unique deadline and a priority. There are two priority levels. Class 1 transactions have higher priorities comparing with class 2 transactions which in turn have priorities higher than class 3 transactions. For transactions within the same class, their priorities will be determined by their deadlines. The transaction with a closer deadline will be given a higher priority.

A transaction is modelled as a sequence of operations. Class 1 and class 3 transactions are modelled as single operation transactions as each update transaction usually only updates one stock. Class 2 transactions are multi-operation transactions. In analysing trading opportunity, it has to access a group of stocks, e.g., to compare the current prices of the stocks in the group to determine which one is most beneficial for trading. Each operation will access one data object. The required data object of an operation is randomly assigned. It means that each data object in the database has the same probability to be accessed by an operation. Although in a database system there may exist some hot-spots, in which the probability of access is higher, it has been found that the major impact of hot-spots on the system performance is a higher lock conflict probability. If the database is small, the effect of hot-spots on the system performance can be studied by using a small database. In the central database site, there is a CPU and a disk which stores the database.

A modified version of H2PL [Abbo88] based on *similarity* is defined for concurrency control. If there is a lock conflict and all the conflicting transactions are *similar*, the lock requesting transaction will be allowed to use the lock and all other lock holding transactions will not be affected. If they are not *similar* and the lock requesting transaction is of higher priority, the lock will be granted and the lock holding transaction will be restarted from its beginning. When a lock is released, the block queue, which contains the blocked transactions due to lock conflicts with higher priority transactions, will be

searched. The highest priority transaction waiting for the lock will be released from the queue. It will join the CPU queue for processing.

If a lock is granted to a transaction, the transaction will access the CPU. After using the CPU, it accesses its required data objects in the disk. The scheduling for accessing the data objects in the disk is also based on their priorities. The granted locks will be released when the transaction is committed, restarted or aborted. A transaction will be allowed to commit if all its operations are processed.

It is assumed that all the transactions are *firm real-time transactions*. Their values will drop to zero once their deadlines are missed. The system continuously checks the deadlines of the executing transactions. Once a transaction is found to have its deadline missed, it will be aborted immediately. Although the update transactions are considered to be firm real-time, they can complete before their deadlines as they are assigned to the highest priorities.

5.2 Implementation of *Similarity*

To simulate the above model using the definition of *similarity* in section 4 requires to keep a value for each data object in the database and also has to define an update value for each update operation. This will make the simulation program very complex and the execution time for the simulator will be very long. So, we have made an assumption to simplify the model implementation. It is assumed that the value of a data object in the database is changed “gradually” from one value to another and there is a maximum rate of change in the value of a data object. For example, if the price of Aloca at present is \$60.0, it is not possible for it to become \$61.0 or \$59.0 in five seconds. The maximum rate of change in price of Aloca is \$0.2/sec. We have checked the validity of this assumption with the situation in the Hong Kong Stock Market. It has been found that in general, it is trueⁱ. With this assumption, the data objects in the database only need to keep locking related information and at which time they are updated. The update transactions only need to carry information about which data objects they want to update and their creation times. The definition of *similarity boundary* is defined in terms of period of time. We assume that the *similarity boundary* for all the data objects is the same.

ⁱ For instance, there are 108 transactions completed on the stock “Citic Pacific” in the stock market on Feb 20 1997 from the period of 11:15:00 to 11:30:00. The stock price of it fluctuates between HK\$38.20 and HK\$38.30. The change is very small and gradual. The reasons of choosing this period are that it is in the middle of the morning section and the stock market is active on that day.

We have designed a time-stamp mechanism for checking *similarity*. Each transaction is assigned a time-stamp which is its creation time. Two additional information is defined for each lock in the lock table. They are :

- (1) Last update time: the arrival time of the last completed update transaction on the data object associated with the lock.
- (2) Earliest request time: the arrival time of the earliest uncompleted transaction holding the lock.

The value to be updated by a lock requesting transaction is similar to the value of the data object in the database if the difference between the last update time of the data object and the arrival time of the lock requesting transaction is smaller than the *similarity boundary* defined for the data. A lock requesting transaction is *similar* with other transactions accessing the same data object if the difference in the arrival time between the earliest request time and the lock requesting transaction is less than the *similarity boundary* defined.

5.3 Transaction Deadline and Performance Measures

Similar to many previous study on real-time concurrency control, the deadline, D_i of a transaction, t_i , is defined based on the expected execution time of t_i :

$$D_i = \text{arrival time of } t_i + \text{slack factor} * \text{expected execution time of } t_i$$

slack factor is a random variable which is uniformly distributed between two bounds: the upper slack bound and the lower slack bound. The slack bounds determine the tightness of the deadline of the transactions. This equation is applied to all the three classes of transactions by assuming that all the transactions have the same tightness of deadlines.

Table 1 lists the some of model parameters and their values used in the simulation experiments. Their values are kept constant for all simulation experiments.

Parameters	Value
Number of data objects in the central database system	200 data objects
Number of sites	3
Number of operations in a class 1 transaction	1

Number of operations in a class 3 transaction	1
Total transaction generation rate for heavy loading	100 Tx/s
Total transaction generation rate for low loading	50 Tx/s
Arrival rate for class 3 transactions under heavy loading in each site	3.3 Tx/s
Arrival rate for class 3 transactions under low loading in each site	2 Tx/s
Upper slack bound	0.5
Lower slack bound	1.5
CPU time to process an operation	3 ms
Simulation run length	1000 sec

Table 1 : Static parameters and their values

The values of some other parameters will be varied in order to test the performance of the model under different workload and system configurations. They are shown in table 2.

Parameters	Value / Value Range Combinations
Arrival rate for class 2 transactions in each site	2, 3, 4, 5, 5.5
Arrival rate for class 1 transactions in each site	Calculate from the arrival rate of class 1, class 2 transactions, and the overall arrival rate of all the transactions
Similar Boundary	0.5s, 1.0s, 1.5s, 2.0s, & 2.5s 1.0s is used when studying other parameters
Number of operations in a class 2 transaction	1 to 9 with average 5 3 to 12 with average 7.5 5 to 15 with average 10 8 to 17 with average 12.5 10 to 20 with average 15 Average 15 is used when studying other parameters
Access time for a data objects in the disk	3ms, 4ms, 5ms, 6ms, 7ms 5ms is used when studying other parameters

Table 2 : Parameters and their values for different experiment settings

The parameter values for the base setting are determined from the simulation results from a set of preliminary experiments. In our simulation model, a small database (200 data objects) is used. The reason of using a small database is that a stock trading company usually will subscribe only a subset of stocks in the stock market. A small database also allows us to study the effect of hot-spots.

For a given system workload, the arrival rate of class 1 transactions is adjusted corresponding to the changes in arrival rate of class 2 transactions. The arrival rate of class 3 transactions is kept constant in all experiments.

The major performance measures used are missing rate, conflict ratio and average block queue length. The block queue contains the blocked transactions due to lock conflicts.

Missing Rate = the number of deadline missing transactions ÷ the number of transactions processed

Conflict Ratio = the number of lock conflicts ÷ the total number of lock requests

Average Block Queue length = the average number of transactions blocked in the block queue

5.4 Simulation Results and Analysis

In the discussion of simulation results, we concentrate on the performance of class 2 and class 3 transactions. Class 1 transactions can always meet their deadlines as their priorities are higher than class 2 and class 3 transactions. The simulator is implemented with the simulation package called OPNET. Each simulation run simulates the time of 1000 sec. With an arrival rate of 100 transactions per second, 100,000 transactions will be completed in each run.

Figure 1 shows the missing rate of class 2 transactions with and without *similarity* as a function of arrival rate of class 2 transactions. The *similarity boundary* is defined to be 1 second. It can be seen that for light system workload (50 Tx/sec), missing rate with *similarity* is consistently much smaller than that without *similarity*. The same observation is obtained with the use of heavy system workload (100 Tx/sec). With *similarity*, lower missing rate is due to smaller number of lock conflicts and restarts. Lock conflicts will result in priority inversion. Transaction restarts will increase the system workload. Both factors decrease the probability of meeting the deadline of the transactions.

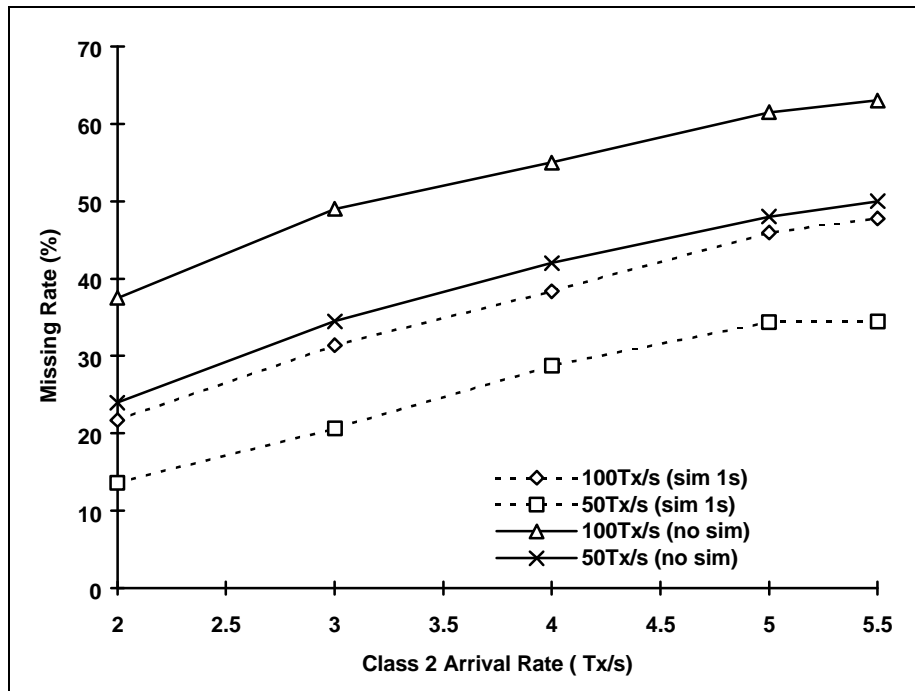


Figure 1 : Missing rate of class 2 transactions with and without similarity

Figure 2 shows the missing rate of class 3 transactions. The results are a little bit surprise. For light system workload (50 Tx/s), the missing rates are very close for the cases with and without *similarity*. For heavy system workload (100 Tx/s), the missing rates with *similarity* are higher than that without *similarity*. It is due to the increase in the number of lock holding transactions as the concurrency of the system is higher due to the use of *similarity*. Consequently, the average lock holding time is increased. Since class 3 transactions are of lower priority as compared with class 1 and class 2 transactions, they are more likely to be restarted or blocked due to lock conflicts. The probability of missing their deadlines is thus increased with the use of *similarity*.

The impact of different *similarity boundary* on the missing rate of class 2 transactions is shown in figure 3. For different arrival rates of class 2 transactions, an increase in the *similarity boundary* can reduce the missing rate. This is especially true for small value of *similarity boundary*. It is because most of the lock conflicts are between transactions with similar arrival times.

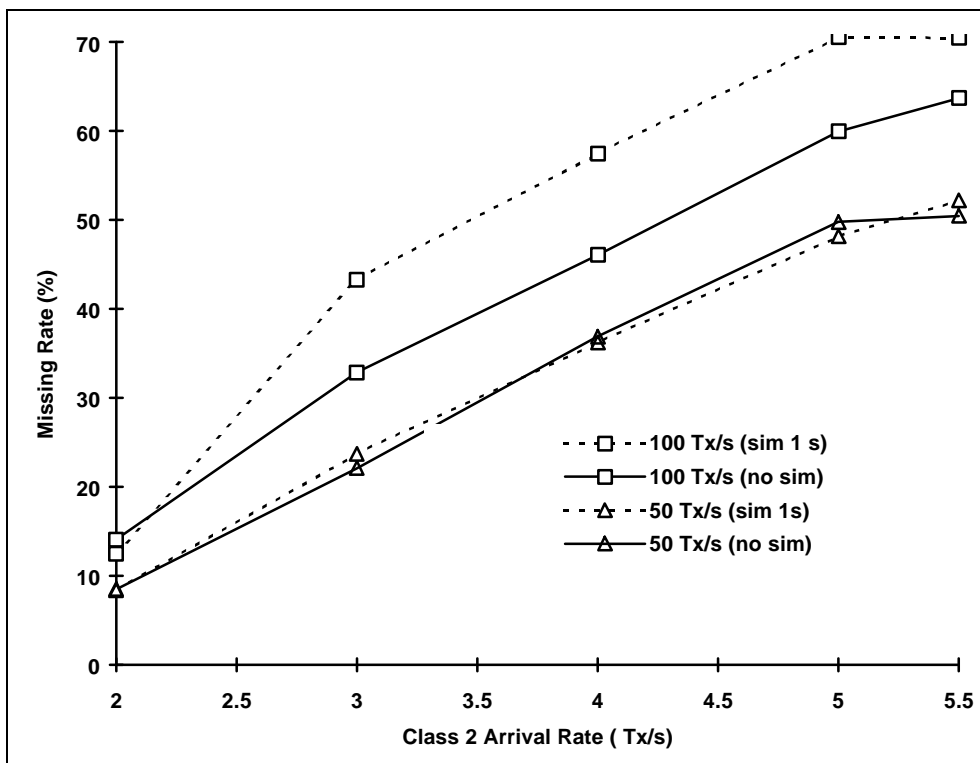


Figure 2: Missing rate of class 3 transactions with and without *similarity*

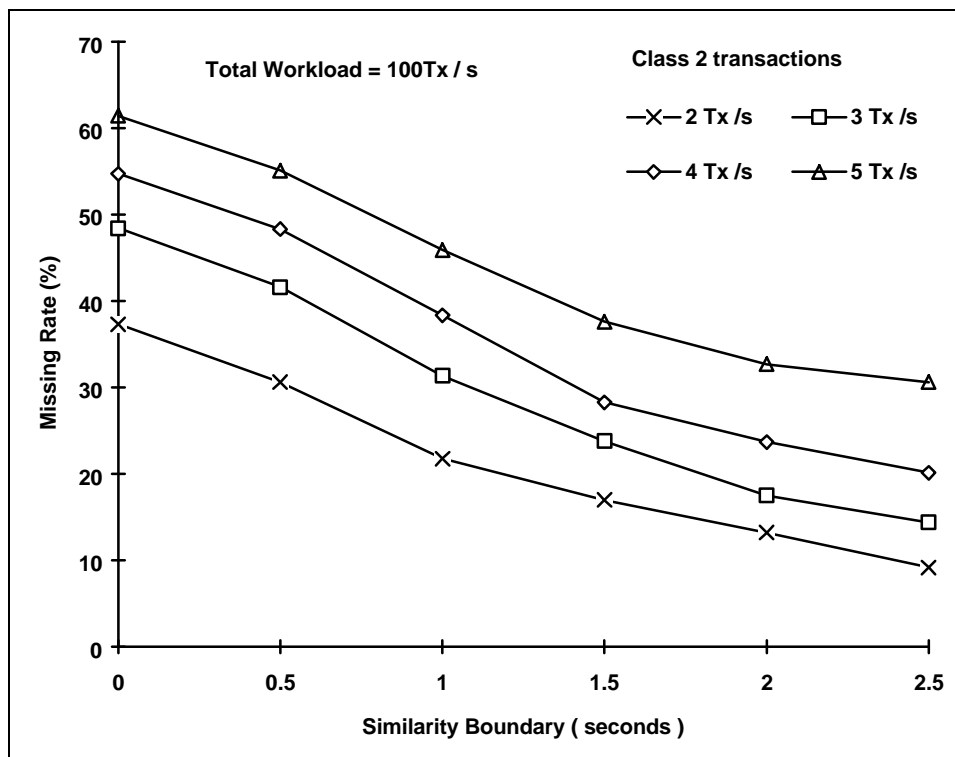


Figure 3 : Impact of *similarity boundary* on missing rate of class 2 transactions

The reason of better performance with an increase in *similarity boundary* is the decrease in the number of lock conflicts and the smaller number of restarts. As can be seen in figure 4, the conflict

ratios drop with an increase in *similarity boundary*. The decrease in number of blocked transactions can be observed in figure 5 which depicts the average length of the blocked queue as a function of *similarity boundary*.

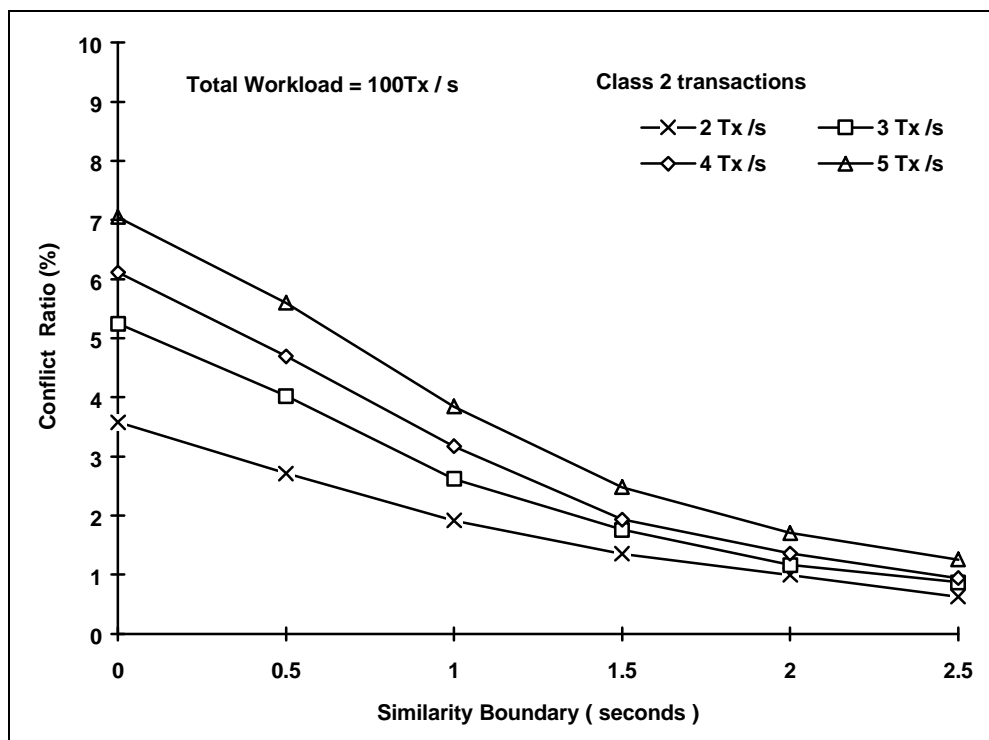


Figure 4 : Impact of *similarity boundary* on conflict ratios

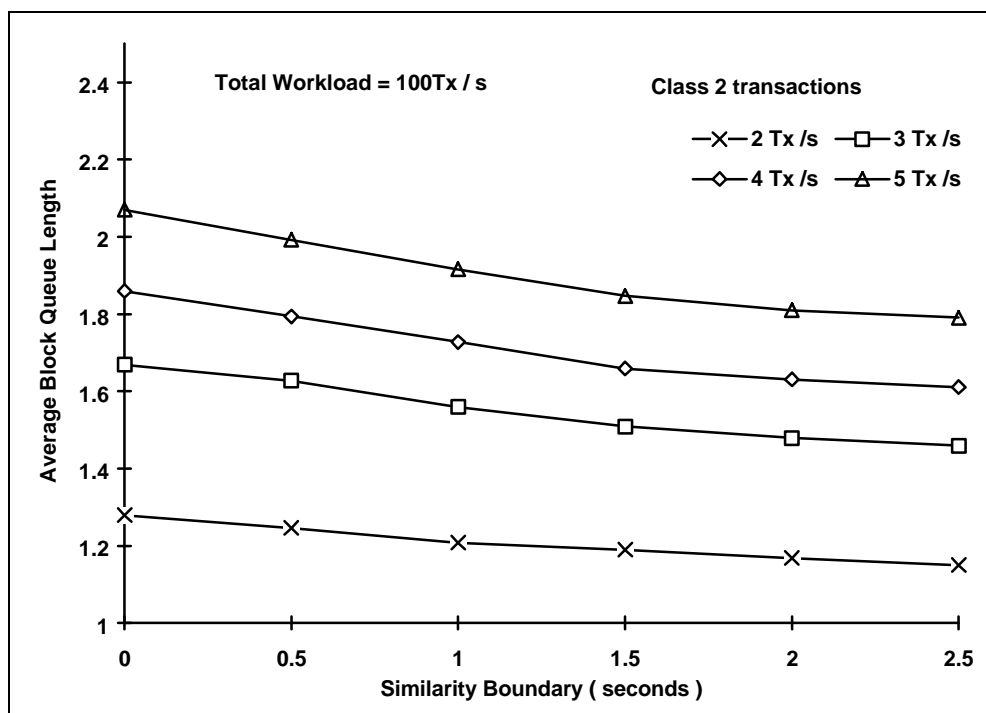


Figure 5 : Impact of *similarity boundary* on average block queue length

To further investigate the performance of the model, we have performed two more sets of simulation experiments. Figure 6 shows the effect of changing the number of operations per transaction on the missing rate. Consistent with the previous results, the missing rate with *similarity* is much smaller than that without *similarity*. The improvement is greater when the number of operations is larger. It is because the probability of lock conflict is higher when the number of operations in a transaction is larger.

Figure 7 depicts the impact of different data access time on the performance of class 2 transactions. A change in access time can be used to model a change in disk speed. The missing rate with *similarity* is also consistently smaller than that without *similarity* for different data access time. The missing rate of both model increases with an increase in access time. It is because the average lock holding time and the average number of lock holding transactions will be increased accordingly if the access time is increased. Both factors increase the probability of lock conflicts and as a result more transactions will be blocked and restarted.

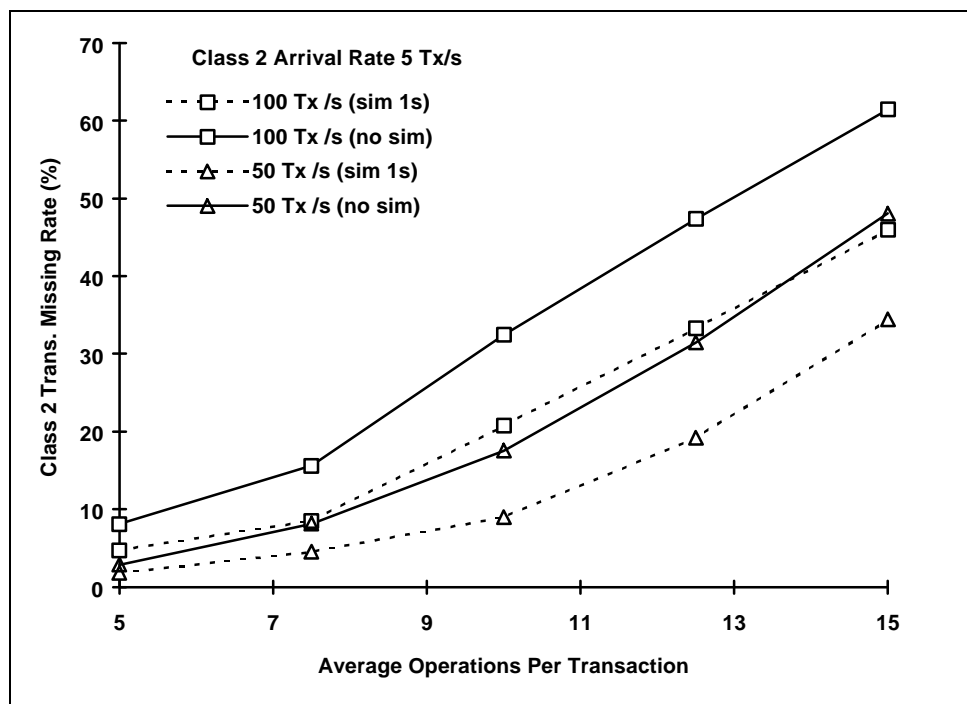


Figure 6 : Impact of changes in number of operations per transaction

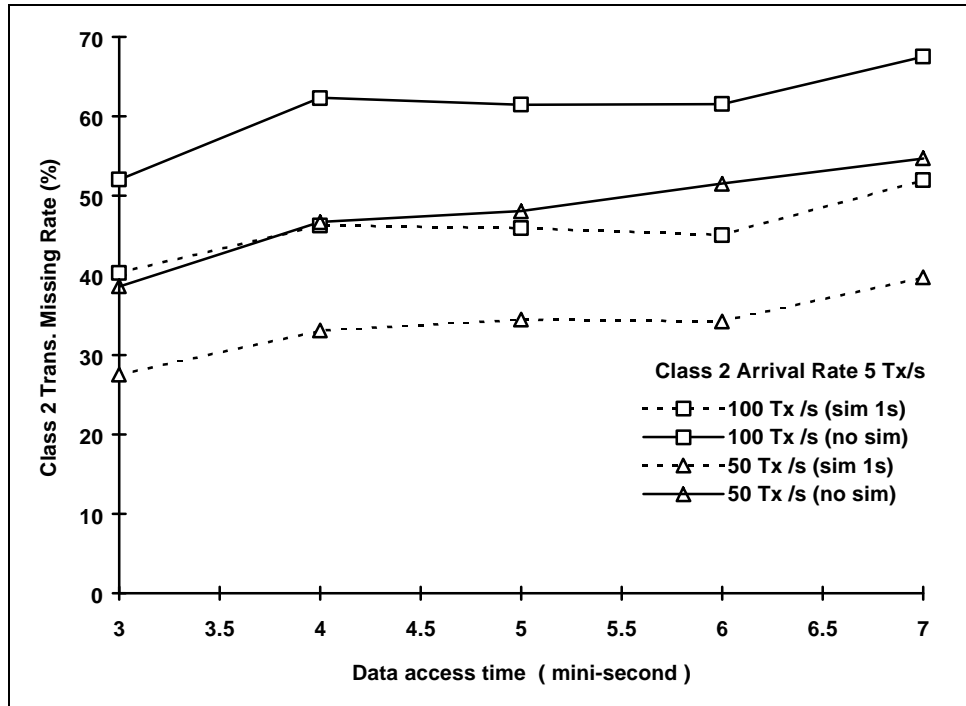


Figure 7 : Impact of data access time on the performance

6 Limitations and Future Directions

In [Kuo92], the concept of *similarity* is proposed as a correctness notion for concurrency control in RTDBS. Although the correctness of similarity have been proven in [Kuo92, Kuo93], only very few of work has been reported on applying *similarity* in real-time database applications. *Similarity* is an application dependent concept. In applying *similarity* in realistic applications, it is likely that a number of practical problems will arise, e.g., how to define a similar data, under which conditions a conflict is considered to be *similar* and how to define concurrency control protocols based on *similarity*. All these questions have to be solved with the use of the semantic requirements of the applications. Up to now, it is completely lack of work to address these issues.

To illustrate the importance and usefulness of *similarity*, the best way is to conduct a real case study. Prior to the implementation of *similarity* in real applications, it will be useful and important if the important issues on the application of *similarity* can be identified first. In this paper, we have discussed how to apply *similarity* for concurrency control in a real-time database application, the stock trading database system. By exploring the requirements and characteristics of the application, we have shown how to define similar data and a concurrency control protocol based on *similarity* for the application. By simulating a stock trading database system, we also have investigated how similarity improves the system performance, and what are the benefits and trade-offs.

Due to the simplifications of the model, such as the assumptions on the workload and the system requirements, not all the problems of applying *similarity* are identified. When it is applied to a realistic real-time database application, other practical problems may arise and the system performance may be different from what we have obtained from simulation. However, it is impossible to development a simulation model without any assumption. Otherwise, the costs to implement and to run the simulation program will be very high and are not affordable. Although these assumptions limit the usefulness of the results, their impact on our findings may not always be substantial. For example, In our model, we have assumed that the inter-arrival time of the update transactions follows exponential distribution and the data access pattern of the transactions is random. If we examine the arrival pattern of the updates in a real life stock trading system, we find that it is highly unpredictable and it is not possible to model with a single distributionⁱⁱ. Similarly, the access pattern of other transactions for retrieving and updating the database is application dependent. In our preliminary experiments, we find that the main impact of different arrival distributions and data access patterns of transactions is on the data conflict probability which has already been examined in the simulation study. Based on the simulation results, we have a much better understanding on *similarity* and in applying *similarity* in realistic real-time database applications. Accompany with the positive findings, it is reasonable to believe that *similarity* is a suitable notion of correctness for concurrency control in real-time database applications and more work should be continued on this direction.

After the simulation study on the application of *similarity* in real-time database applications, the next step on the research of *similarity* for RTDBS is to develop a real-time database prototype with *similarity* and to implement a realistic real-time application using the prototype. With the prototype, lesser assumptions need to be made. The design and implementation of the prototype can follow some real systems. By driving the prototype with real test workload, which can be extracted from real life applications, more characteristics and properties of *similarity* can be identified. Another way to test the prototype is to use benchmark programs. However, up to now, there is no representative benchmark program for this kind of applications. The design and development of new benchmark programs will be an important future work to the research on RTDBS.

In order to provide a better support to real-time database applications, another direction of the research on the area is to add active properties to RTDBS such as the capability to trigger transactions under certain conditions. In triggering transactions, the relationship amongst the triggering and the

ⁱⁱ We have examined the arrival pattern of updates in the Hong Kong Stock market. We find that it is not possible to define a simple distribution to describe the pattern. The arrival rate is dependent on many factors and is highly time dependent. For instance, the number of transactions is larger at the end of a day than at the start of a day.

triggered transactions is important to the system performance. At the same time, this relationship will affect the design of concurrency control protocols. The investigation on the performance of concurrency control protocols based on *similarity* under this new framework will be useful to the system designers and is an area worth for further study.

7 Conclusions

Concurrency control is an important issue in the design of real-time database systems (RTDBS). In the last few years, a number of real-time concurrency control protocols have been proposed. The notion of correctness adopted in these protocols is serializability which is too restrictive and is not suitable for many RTDBS. Owing to the unique characteristics of RTDBS and their applications, a less restrictive notion of correctness should be used. One recent suggestion is *similarity* which releases the serializability constraint by the definition of similar data. However, how to apply the concept of similarity in real-time database applications is still not clear. Also, it is lack of study on how similarity improve the performance of a RTDBS.

In this paper, *similarity* is used as the correctness notion for concurrency control in a RTDBS, a stock trading database system. In a stock trading database system, there exists different kind of transactions with different characteristics and priorities. By exploring the properties and requirements of the transactions, the real-time concurrency control protocol, H2PL, is re-defined using *similarity* as the correctness notion. A simulation model of the system has been implemented to investigate how *similarity* improves the system performance. It has been found that *similarity* can significantly reduce the number of data conflict and thus increases the degree of concurrency. The number of deadline missing transactions can be much reduced. Even if the *similarity boundary* is very small, a very significant improvement in performance can be obtained. The positive results on *similarity* confirm our belief that *similarity* is a more suitable notion of correctness for concurrency control in RTDBS.

References

- [Abbo88] Abbott, R.J. and H. Garcia-Molina, "Scheduling Real-Time Transactions : A Performance Evaluation", in Proceedings of the 14th VLDB Conference, 1988, pages 1-12.
- [Adel95] B. Adelberg, H. Garcia-Molina, and B. Kao, "Applying Update Streams in a Soft Real-time Database System", in Proceedings of ACM SIGMOD Conference, 1995, pages 245-256.
- [Agra95] D. Agrawal, El Abbadi, R. Jeffers and L. Lin, "Ordered Shared Locks for Real-time Databases", The VLDB Journal, volume 4, number 1, pages 87-126, 1995.

- [Bern87] Bernstein, P.A., Hadzilacos, V., & Goodman, N., "Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading, Mass., 1987.
- [Best95] Azer Bestavros and Spyridon Braoudakis, "Value-cognizant Speculative Concurrency Control", in Proceedings of International Conference on Very Large Data Base 1995, 1995.
- [Best96] Azer Bestavros, "Advances in Real-time Database System Research", ACM SIGMOD Record, volume 25, number 1, 1996.
- [Eswa76] K.P. Eswaran, J.N. Gray, R.A. Lorie and I.L. Traiger, "The Notions of Consistency and Predicate Locks in Database System", Communications of the ACM, volume 19, number 11, pages 624-633, 1976.
- [Garh94] Marc H. Garham, "How to get serializability for real-time transactions without having to pay for it", in Proceedings of Real-Time Systems Symposium 1993, pages.56-65.
- [Gray93] J. Gray and Reuter, A., "Transaction Processing : Concept and Techniques", Morgan Kaufmann, 1993.
- [Hari92] J.R. Haritsa, M.J. Carey and M. Livny, "Data Access Scheduling in Firm Real-time Database Systems", Journal of Real-time Systems, volume 4, number 3, pages 203-242, 1992.
- [Huan92] J. Huang, J.A. Stankovic, K. Ramamritham and D. Towley, "Priority Inheritance in Soft Real-time Databases", Journal of Real-time Systems, volume 4, number 2, pages 243-268, 1992.
- [Kim95] Young-Kuk Kim and Sang H. Son, "Predictability and Consistency in Real-time Database Systems", in Advances in Real-time Systems, edited by Sang H. Son, pages 509-531, Prentice Hall, New York, 1995.
- [Kuo92] Tei-Wei Kuo and Aloysius K. Mok, "Application Semantics and Concurrency Control of Real-Time Data-Intensive Applications", in Proceedings of IEEE 13th Real-time Systems Symposium, 1992, pages 35-45.
- [Kuo94] Tei-Wei Kuo and Aloysius K. Mok, "Using Data Similarity to Achieve Synchronisation for Free", in Proceedings of 11th IEEE Workshop on Real-Time Operating Systems and Software (RTOS), 1994, pages 112-116.
- [Lam95] K.Y. Lam and Hung, S.L., "Concurrency Control for Time-constrained Transactions in Distributed Database Systems", The Computer Journal, volume 38, number 9, pages 704-716, 1995.
- [Lee94] J. Lee, "Concurrency Control Algorithms for Real-time Database Systems", Ph.D. Thesis, Department of Computer Science, University of Virginia, 1994.
- [Lin89] K.J. Lin, "Consistency Issues in Real-time Database Systems", in Proceedings of 22nd Hawaii International Conference on System Science, pages 654-661, 1989.
- [Liu73] C.L. Liu and Layland, J.W., "Scheduling Algorithms for Multi-programming in a Hard-Real-Time Environment", Journal of the ACM, volume 20, number 1, pages 46-61, 1973.
- [Ozso95] G. Ozsoyoglu and R. Snodgrass, "Temporal and Real-time Databases: A Survey", IEEE Transactions on Knowledge and Data Engineering, volume 7, number 4, pages 513-532, 1995.
- [Rajk91] R. Rajkumar, "Synchronisation in Real-time Systems: A Priority Inherent Approach", Kluwer Academic Publishers, The Netherlands, 1991.

- [Rama93] K. Ramamritham, "Real-time Databases", International Journal of Distributed and Parallel Databases, volume 1, number 2, pages 199-226, 1993.
- [Rama95] K. Ramamritham, "The Origins of TCs", in Proceedings of International Workshop on Active and Real-time Database Systems, June 9-11 1995.
- [Shu92] Shu LihChyun and Michal Young, "Correctness Criteria and Concurrency Control for Real-Time Systems: A Survey", Software Engineering Research Center, Department of Computer Sciences, Purdue University; 1992.
- [Stan95] J. Stankovic, Spuri, M. and Natale, M.D., "Implications of Classical Scheduling Results for Real-time Systems", IEEE Computer, volume 28, number 6, pages 16-25, 1995.
- [Son88] S. H. Son, "Real-time Database Systems: Issues and Applications", ACM SIGMOD Record, volume 17, number 1, pages 2-3, March 1988.
- [Ulus94] O. Ulusoy, "Processing Real-Time Transactions in a Replicated Database System", Journal of Distributed and Parallel Database, volume 2, number 4, 1994, pages 405-436.
- [Yu94] P.S. Yu, Wu, K.L., K.J. Lin and S.H. Son, "On Real-time Databases: Concurrency Control and Scheduling" Proceedings of IEEE, volume 82, number 1, pages 140-157, 1994.