Locking Protocols for Concurrency Control in Real-time Database Systems

S.L. Hung and K. Y. Lam
Department of Computer Science
City Polytechnic of Hong Kong
CSHUNG@CPHKVX.BITNET

ABSTRACT

Concurrency Control in real-time database systems is complicated by the requirement to maintain database consistency at the same time to minimize the number of transactions missing their deadlines. The scheduling of data in Basic Two Phase Locking (2PL) completely ignores the urgency of a transaction and thus the effectiveness of the adopted real-time resource scheduling protocol is greatly reduced. In Restart based locking protocols (R2PL), same priorities are used for both data and resources scheduling and should have fewer transactions missing their deadlines. However, Restart based protocols sufferred the intrinsic weakness of high restart overhead owing to ensure atomicity of transactions. In this paper, based on their weaknesses, a hybrid concurrency control protocol (H2PL) is proposed. Through performance study, results indicate that it can perform well under different degree of deadline constraint and workload as compared with other real-time locking protocols.

1. INTRODUCTION

Current trends indicated that there is an increasing demand for real-time database systems in which transactions have explicit timing constraint. The value of a transaction is severely affected if it cannot complete within its time frame. In this paper, the focus is on hard real-time systems in which the 'value' of a transaction will be completely lost if it cannot complete before its deadline. There are many new and challenging problems associated with the design of such kinds of systems. One of them is the scheduling of transactions. In contrast to other real-time systems, there are two types of scheduling in real-time database systems, hardware and data. For hardware resources scheduling, the timing constraint of a task is satisfied by different priority driven scheduling algorithms on hardware resources, i.e. CPU. Priority are assigned to different tasks based on its certain information such as criticalness, deadline and slack time. Preemption is usually allowed in order to prevent long delay of urgent tasks. Some of the efficient real-time scheduling methods are least slack time first, earliest deadline first (EDF) and rate monotonic [4,5,7].

The scheduling of data is required for maintaining the consistency of database. Different concurrency control protocols have been proposed. Traditionally, they are based on serializability theory which does not allow preemption of sharing data except when it is accompanied with the corresponding undo operations to restore the state of database. One of the most commonly adopted concurrency control method is the locking protocols. These protocols can be classified into blocking and restart based. Both types of protocols have been proven in maintaining database consistency. However, they pay no attention to the timing constraint of transactions and may affect resource scheduling of transactions.

In blocking based protocols, high priority transactions may be blocked by low priority transactions due to lock conflicts. This blocking time is unbounded as chain of blocking may form and repeat blocking is possible. Furthermore, priority inversion [7] may occur when the lock seizing transaction has a lower priority. In order to resolve its intrinsic weaknesses, different real-time concurrency control protocols have been proposed [1,4,5]. In [1], Abbott and Molina have proposed a restart based Two Phase Locking protocol (R2PL). It incorporates priority information in lock setting so as to minimize the impact of the concurrency control protocol on the scheduling of hardware resources. Transactions with higher priority in hardware resources will be given a preference in setting of locks. Whenever a higher priority transaction is in conflict with a lower a priority transaction, the lower priority transaction will be aborted and restarted later on. The conflicting lock is granted to the higher priority one. One of its major weaknesses is that it neglects the impact of restart on the scheduling of other transactions. Restarting a transaction is very costly especially under high degree of resource contention [2,3]. A large number of restarts increase the workload of the whole system and may cause some other coming transactions to miss their deadlines.

To reduce the number of restarts, Abbott and Molina [1] also suggested the conditional restart protocol in which lower priority transaction will have to be restarted only if the slack time of the higher priority transaction is smaller than the remain execution time of the holder. However, the effectiveness of this checking is greatly affected by the probability of blocking. Furthermore, priority inversion and deadlock is still possible.

In this paper, a hybrid real-time concurrency control protocol (H2PL) is proposed. The resource scheduling part of the protocol is based on EDF and the scheduling of data is a hybrid of blocking and restart. The performance of the protocol is compared with the other two protocols.

2. THE HYBRID REAL-TIME CONCURRENCY CONTROL PROTOCOL (H2PL)

The H2PL protocol is based mainly on the conditional restart protocol as proposed by Abbott and Molina [1]. Some of the additional features in the new design include checkings on certain conditions to avoid unnecessary restarts, prevention of priority inversion by enforcing priority inheritance [6] and limiting the blocking to one transaction (prohibiting chain to form). Transactions (requester, T_r) is blocked only if it is not blocking any other transaction or the lock owner (holder, T_h) is not being blocked by another transaction. In case of chain forming, conflict is resolved by aborting the lower priority transaction. (Aborting the higher priority transaction may results in cyclic restart as it is the next one to execute.) The detail of the protocol is as follow :

Whenever there is a lock conflict, the priority of the requesting transaction and holder transaction will be checked. Based on the result of evaluation, the following action will be done.

CASE :

```
(1) P(Tr) = P(Th)
        Abort(Tr);
        Restart(Tr)
```

where P(T) is the priority of transaction T. Note Abort(T) means to stop process transaction T and performs its undo operations. Restart(T) is restarting transaction T from its beginning.

***** It is assumed that the deadline of each transaction is unique. Therefore, case (1) is needed so as to prevent the formation of deadlock cycle, i.e. the holder is blocked by the requester and the priority of the requester has inherited the priority of the holder. (A condition which does not exist in Abbott & Molina's algorithm [1].)

```
(2) P(Tr) > P(Th)
      IF  S(Tr) < R(Th)                    .....(a)
        IF  S(Th) < (E(Th) + R(Tr))  .....(a)
            IF  Bk(Th)                     ...(b)
                P(Th) <= P(Tr)             ..(d)
                Abort(Th)
            ELSE
                IF  PC(Tr) > PC(Th)    ..(b)
                    P(Th) <= P(Tr)  .(d)
                    Abort(Th)
                ELSE
                    Abort(Tr)
                ENDIF
            ENDIF
        ELSE
            P(Th) <= P(Tr)                 ...(d)
            Abort(Th);
            Restart(Th)
        ENDIF
      ELSE
        IF  Bk(Th)                         ....(c)
            P(Th) <= P(Tr)
            Abort(Th);
            Restart(Th)
        ELSE
            IF  Bk(Tr)                     ...(c)
                P(Th) <= P(Tr)  ..(d)
                Abort(Tr);
                Restart(Tr)
            ELSE
                P(Th) <= P(Tr)  ..(d)
                Block(Tr)
        ENDIF
      ENDIF
    ENDIF;
```

where S(T), E(T) and R(T) is the slack time, expected total execution time and remain expected execution of transaction T. Bk(T) is a checking on transaction T whether it is being blocked or blocking other transactions. Block(T) puts transaction T into the block queue. PC(T) is the calculation of the proportion of completion for transaction T. That is the proportion of work performed over its initial total service requirement.

***** Case (2), the words in italic are the original conditional restart protocol.

(a) : If either the waiting requester or restart of the holder misses its deadline, one of the them has to be selected for abortion.

(b) : If the holder is being blocked, it will be selected for abortion else the transaction with the smaller workload will be selected for abortion (by comparing the proportion of completion for each).

(c): Chain of blocking is prevented as the blocking of requester is allowed only after checking that the requester and holder is not being blocked or blocking other transactions respectively.

(d) : Whenever a lower priority transaction has to be restarted, its priority will be raised to that of the requesting transaction so as to remove

the possibility of priority inversion. The priority of the restart transaction will be restored to its initial value after the completion of its undo operations and the release of its locks.

(3) $P(T_r) < P(T_h)$

```
IF PC(Th) > PC(Tr)
    Abort(Tr);
    Restart(Tr)
ELSE
    Abort(Th);
    Restart(Th)
ENDIF;
```

***** Case (3) is required in the case when the holder is being blocked. (If not, it should be executing as it has the highest priority.) One of the transactions have to be aborted and restarted.

ENDCASE.

3. PERFORMANCE MODEL

The database model is assumed to be main memory resident [8] for two reasons. Firstly, it is always argue that real-time database systems should be main memory resident in order to support a fast I/O response time. Secondly, having only one resource in the model allows us to concentrate on the impact of concurrency control on the scheduling of resources. In the model, it is assumed that for each transaction, the deadline and expected execution time is known upon its arrival. In addition, its remain execution time is kept track by the system. All this information is important for the determination of its eligibility and its priority for execution. The eligibility for execution is based on feasible deadline as this has been shown to have the best performance [1]. Under this method, before the execution of a transaction, its deadline will be checked again with its remaining expected execution time. If it is found that it is not feasible to complete before its deadline, the transaction will be immediately decided to abort so as to minimize the amount of waste work. An aborted transaction is terminated after the completion of its undo operations and releasing of its seized locks. The model in [1] has neglected the overhead for lock management which has been shown to be significant when the degree of data contention is high [9]. The lock table is considered to be a critical region. No preemption is allowed during lock setting and releasing.

Figure (1) depicts the model for our real-time database system. Transactions will be continuously being generated. The inter-arrival time is assumed to be exponentially distributed. Each arrived transaction is ready for execution and will be put into the ready queue which queuing discipline is priority on deadline. Whenever a transaction has finished its execution, the scheduler will select the head of the ready queue for execution. If a new job arrived, the scheduler will compare the priority of the one currently executing with the new job. It selects the higher priority job to execute and put the lower one into the ready queue. The execution of a transaction is an alternating sequence of lock setting, data access and computation. If the lock request is rejected due to lock conflict, the requesting transaction will be suspended and put into the block queue. It stays in the queue until its requesting lock is released by the seizing transaction. If there is any updates, the original value written into the transaction log first to cater for the problem recovery. After all the computation of a transaction have been performed, all its seized locks will be released at once upon its commitment.

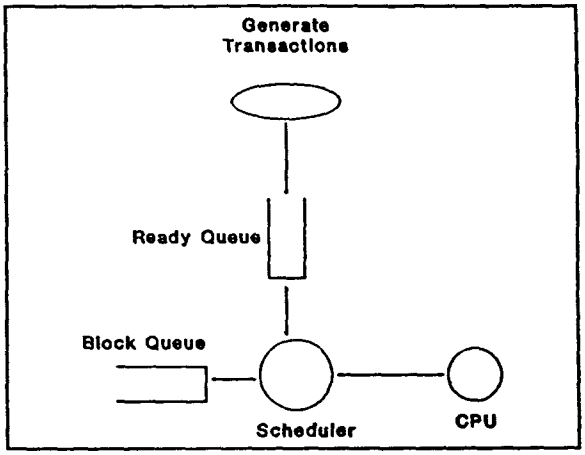

Figure (1) : Real-time Database Model

4. MODEL PARAMETER AND PERFORMANCE MEASURES

The protocols are tested under different workload and deadline constraints. The deadline of a transaction is depended on the expected execution time of it and a slack variable which is a random variable follows exponential distribution. The selection of the mean value for generating the variable is uniformly distributed between the maximum and minimum slack.

For example, the deadline is calculated as :

slack factor = Uniform(max_slack, min_slack)
slack variable = Exponential(slack factor)
Deadline = Current time + (total expected execution time * slack variable

A wide range of slack time is chosen to represent

different of deadline constraint. The chosen values are :

max & min slack used : 0.5 - 5 (tight deadline)
2.5 - 7 (loose deadline)

One of the greatest problem in performance study is to choose relevant parameters values which are able to demonstrate the performance characteristics of the protocols under studied. In order to be able to observe interesting performance characteristics without unduly long simulation run, a high probability of lock conflicts among different transactions is highly desirable. The overhead of log management is usually small compare with the processing time. With the above considerations, the following base values of parameters are chosen :

No. of data object in the database : 100
Mean transaction size : 15 data objects
Mean CPU time to process a data object : 30 ms
Mean CPU time to check a lock : 1 ms
Mean CPU time to set a lock : 1 ms
Mean CPU time to release a lock : 2 ms
Mean CPU time to create an update log : 6 ms
Mean CPU time to undo an update : 6 ms

Table (1) : Model Parameters and their values

The performance of the concurrency control protocols in real-time database are compared by calculating the missing ratio and the relative missing ratio (RMR) which is defined as :

$$RMR = \frac{missing\ ratio}{missing\ ratio\ for\ the\ base\ model}$$

where missing ratio is the mean proportion of transactions missing their deadlines in each set of simulation runs. In our simulation experiments, each set consists of 6 runs and each run containing 1000 terminated transactions. The control model is the one with no data contention and thus the impact of concurrency control on the real-time scheduling protocol is zero. The mean ready queue length and block queue are also measured to quantify the degree of resource and data contention in the system respectively.

5. PERFORMANCE RESULTS AND INTERPRETATION

In this section, we are going to discuss two of our experiment results. The first one assumes each transaction is belonging to the same class. Their deadline follows similar distribution and their criticalness are the same. The second one assumes there are two types of transactions and they are difference in their criticalness and deadline distributions.

Single Transaction Class

The concavely shaped relative missing ratio (RMR) curves shown in figure (2) indicates that the impact of concurrency control protocols are relatively more important at medium degree of workload. For low workload, the degree of both resource and data contention is small, different concurrency control protocols can achieve similar results. For heavy workload, the impact of concurrency control protocols is less significant as the resource scheduling of hardware is already very tight.

Among the three protocols tested, RMR value is the smallest for H2PL and the greatest for R2PL. The poor performance for R2PL is due to high degree of resource contention. As shown in figure (3), its Ready Queue length is much higher than 2PL and H2PL. The non-zero mean Block Queue length for R2PL in figure (4) is due to the higher priority requester still has to wait for the lower priority holder to perform undo operations. The shortest Ready Queue length for 2PL as shown in figure (3) is due to blocking effect. Many transactions are blocked in the blocked queue and the degree of resource contention is thus much smaller than the other two. This blocking effect is harmful only when the highest priority transaction is being blocked and cannot be scheduled. Since H2PL is based on a hybrid of blocking and restart, its lengths of Ready Queue and Block Queue are between that of the 2PL and R2PL as can be observed from figures (3) and (4).

The result of using a less constrained deadline is shown in figure (5). It can be seen that it is similar to figure (1) except that the difference in their performance is greater. A lesser constraint in deadline reduces the impact of blocking as the slack time of transaction is always large enough to tolerate some degree of blocking. Therefore, many restarts are not necessary.

Mixed Transaction Class

Although R2PL is able to reduce the impact of concurrency control on high priority transaction, at the

same time, it affects all other coming transactions by increasing the workload of the system. If there are two types of transactions in the system and they are different in their criticalness and deadline constraints, the use of R2PL may be desirable if we want to improve the performance of the more critical transactions which has a tighter deadline constraint. Figures (6) and (7) depict the result for them. Compared with 2PL, the number of committed less critical transactions in R2PL is smaller as can be observed in figure (6). Its smaller value is due to the preference to more critical transactions. Less urgent transactions are more likely to be preempted and restarted. They have higher probabilities of missing their deadlines. On the other hand, the performance of more critical transactions is better in R2PL when compared with 2PL especially under heavy workload, as shown in figure (7). H2PL tries to give preference to more urgent transactions, at the same time to minimize this impact on less critical transactions. Therefore, the performance of less critical transactions are similar to 2PL except under heavy workload as can be observed in figure (6). When degrees of both resource and data contention are high, the probability of restarting and thus impact on the resource scheduling protocol will be significant. However, the performance of more critical transactions is much better, especially for heavy workload.

6. CONCLUSION

In this paper we have discussed the problems of blocking and restart based locking protocol for concurrency control in real-time database systems. A hybrid concurrency control protocol is proposed to address their weaknesses. From our performance experiments, we shown that its impact on resource scheduling protocol is smallest and its performance is superior to restart or blocking based real-time locking protocols and is less sensitive toward changes in system workload and deadline constraints.

The problem associated with restart locking protocol in a single transaction class system is that a heavy workload may be created and the overall system performance will be degraded as a result. However, for mixed transactions systems with each class of transaction having different criticalness and deadline constraints, R2PL may be more desirable than 2PL as it improves the system performance of the more critical transactions at the expense of less critical transactions. H2PL has a good performance for different types of transactions because it is capable of giving preference to the scheduling of data for high priority transaction at the same time minimizing this impact on lower priority transactions.

REFERENCES

[1] R. Abbott & H. Garcia-Molina, "Scheduling Real-time Transactions: a Performance Evaluation", Proceedings of the 14th VLDB Conference, 1988, pp.1-12.

[2] R. Agrawal, J.M. Carey & M. Livny, "Concurrency Control Performance Modeling: Alternatives Implication", ACM Transactions on Database Systems, Volume 12, Number 4, pp.609-654.

[3] P.A. Bernstein, V. Hadzilacos & N. Goodman, "Concurrency Control and Recovery in Database Systems", Addison-Wesley, 1987.

[4] A.P. Buchman, D.R. McCarthy, M. Hsu & U. Dayal, "Time-Critical Database Scheduling: A Framework for Integrating Real-time Scheduling and Concurrency Control", Data Engineering Conference, February 1989, pp.470-480.

[5] J. Huang, J.A. Stankovic, D. Towley & K. Ramamritham, "Experimental Evaluation of Real-time Transaction Processing", IEEE Real-time Systems Symposium, 1989, pp.144-153.

[6] L. Sha, R. Rajkumar & J.P. Lehoczky, "Concurrency Control for Distributed Real-time Databases", ACM SIGMOD Record, Volume 17, Number 1, March 1988, pp.82-98.

[7] S. Davari & L. Sha, "Sources of Unbounded Priority Inversions in Real-time Systems and a Comparative Study of possible Solution", 1992, pp.110-120.

[8] M. Singhal, "Issues and Approaches to Design of Real-time Database Systems", ACM SIGMOD Record, Volume 17, Number 1, March 1988, pp.19-33.

[9] P. Franaszek, J.T. Robinson & A. Thomasian, "Concurrency Control for High Contention Environment", ACM Transactions on Database Systems, Volume 17, Number, June 1992, pp. 304-305.

Figure (2) : Relative Missing Ratio (RMR) for R2PL, 2PL & R2PL

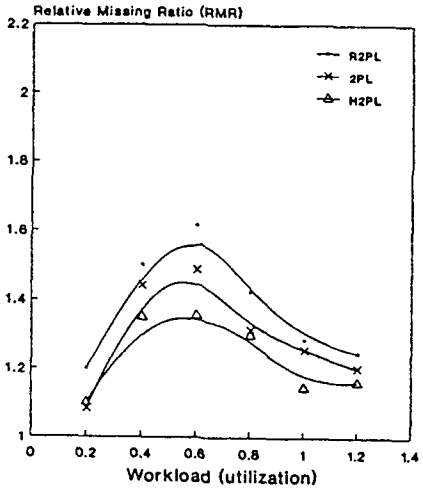

Figure (3) : Mean Ready Queue Length for R2PL, 2PL & H2PL

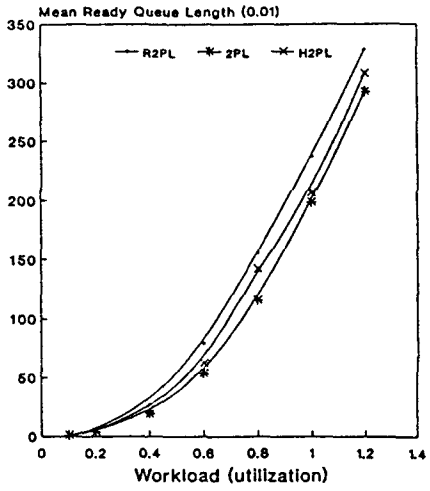

Figure (4) : Mean Block Queue Length for R2PL, 2PL & H2PL

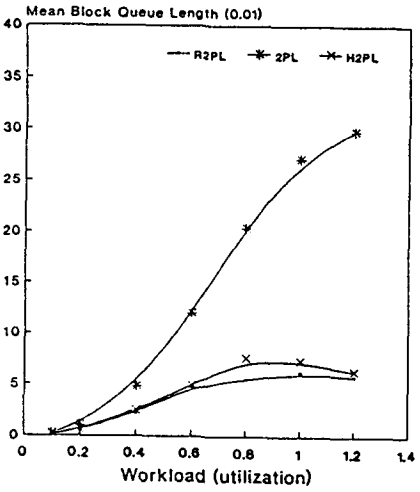

Figure (5) : Relative Missing Ratio for R2PL, 2PL & H2PL under loose deadline contraint

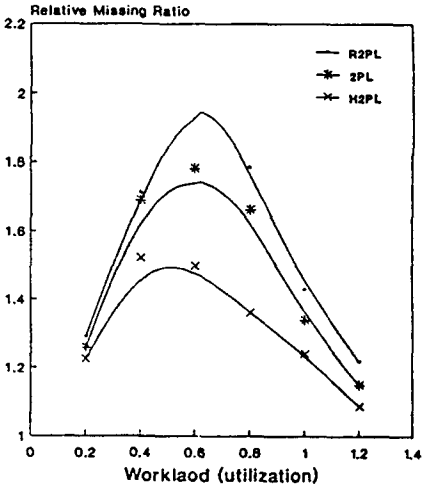

Figure (6) : Missing Ratio for 2PL, R2PL & H2PL (Lower Priority Transactions)

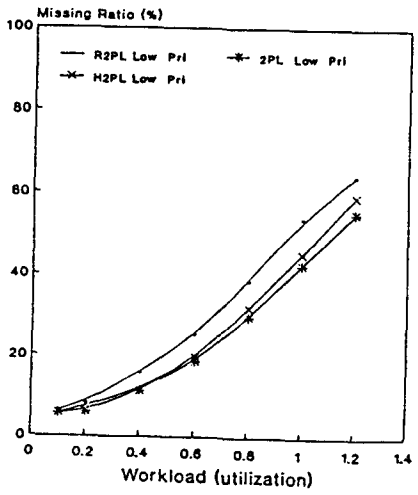

Figure (7) : Missing Ratio for 2PL, R2PL & H2pl (Higher Priority Transactions)