Later Validation/Earlier Write: Concurrency Control for Resource-Constrained Systems with Real-Time Properties

Kamal Solaiman, Graham Morgan
School of Computing Science
Newcastle University
Newcastle-upon-Tyne, UK
{Kamal.Solaiman, Graham.Morgan}@ncl.ac.uk

***Abstract*— We describe an algorithm for optimistic concurrency control suitable for governing transactions operating on databases residing on resource constraint devices. We are concerned with real-time applications that utilize such devices in a computationally demanding manner (e.g., gaming). Therefore, increasing energy efficiency and reducing latency are primary goals for our algorithm to afford higher overall performance and longevity of battery life. We attempt to improve energy efficiency by reducing persistent store access and satisfy real-time requirements via transaction scheduling that affords greater determinism.**

Keywords-component; transaction; serializability; real-time database system; optimistic concurrency control

I. INTRODUCTION

Concurrency control is a mechanism for coordinating simultaneous access to shared data. In databases such access occurs when more than one transaction at a time reads or writes to the same data. In such a setting, the main goal of a concurrency control algorithm is the creation of an ordered read/write schedule that ensures database correctness (atomicity, consistency, isolation, durability - ACID). Two approaches have been investigated in the literature: (i) pessimistic [1]; (ii) optimistic [2].

In a pessimistic approach transactions are blocked (via the mutual exclusion/locking of data) until such a time that there is no possibility that an incorrect schedule may be created. In optimistic concurrency control blocking is reduced; transactions are allowed to proceed but there is a possibility that one or more transactions are rolled back to satisfy correctness (correcting errors in the schedule via pre-commit validation). To allow the optimistic approach, writes are held local to the transaction until the schedule is deemed correct and a transaction can commit.

Optimistic approaches have the potential to provide greater performance than pessimistic approaches. This occurs when the locking incurred in pessimistic approaches inhibits transaction progression more than the cost of rolling back only those transactions that violate correctness. As a general rule, the lower the contention for shared data the more appropriate the optimistic approach becomes.

The rollback and restart of transactions is not usually viewed as a great cost to a system's resources if the only other option is a worse performing pessimistic approach (i.e., more waiting/blocking). However, if limiting persistent state access is considered important then this may negate these performance gains. Such a situation may occur if power consumption is at a premium (as persistent access is usually more expensive than local memory access). This is a common occurrence for mobile devices.

In this paper we present an algorithm that affords the benefit of optimistic concurrency control while limiting persistent state access that is a result of transaction rollback. Considering platforms with limited power and high persistent store latencies inspires our work. More specifically, gaming devices with hardware update cycles measured by the decade. Classic examples are gaming devices such as the Sony PSP, and Nintendo DS. In such systems any advantage that can be gained in terms of performance is desirable. In addition, applications that run on such devices are typically real-time in nature and exhaust all available resources if possible (e.g., if graphics detail can be increased it will be increased). This adds another constraint, that of determinism in the run-time schedule that we wish to consider.

II. BACKGROUND AND RELATED WORK

In this section we provide a brief description of basic optimistic concurrency control to allow the reader to understand our algorithm. This basic description continues with advancements that have been made in the validation process for optimistic concurrency control. We then describe how the technique of re-running aborted transactions from in-memory values has afforded increases in overall system performance. Finally, we identify the contribution of our algorithm by utilizing the described techniques in such a manner as to satisfy our own requirements of increased determinism coupled with limiting access to persistent store.

A. Optimistic Concurrency Control

Kung and Robinson proposed the optimistic approach via a three phased transaction execution (shown in figure 1) [2].

Figure 1. OCC algorithms phases

During the read phase transactions access data without restriction and make their own private copies of such data. All computation that is carried out by a transaction occurs on the private copy. When a write is requested it is enacted on the private copy. During the validation phase a resolution policy is enacted. In principle, other executing transactions are considered to determine if the write requests can be satisfied without invalidating the correctness of the overall read/write schedule. If the writes are valid, the write phase is enacted that commits the changes to persistent storage. Alternatively, the transaction may abort if a valid schedule is not possible and tries again later. If a transaction has no writes, then the write phase is not required, with commit being enacted to bring the transaction to a logical end.

B. *Forward and Backward Validation*

Harder proposes two schemes [3]: Backward Oriented Optimistic Concurrency Control (BOCC) and Forward Oriented Optimistic Concurrency Control (FOCC):

- *BOCC* – intersection between the read set of a validating transaction is compared with the write sets of currently executing transactions. An assumption is made that these other transactions finished their read phases before the validating transaction.
- *FOCC* – intersection between write set of validating transaction and the read sets of currently executing transactions that have yet to finish they're read phase are compared.

In BOCC intersections will require the validating transaction to abort. In FOCC there is a degree of flexibility in that a number of strategies are possible [3]:

- Delay the validating transaction and restart the validation phase later.
- Abort all conflicted transactions and commit validating transaction.
- Abort validating transaction only.

FOCC has found popularity with researchers due to the flexibility it affords in terms of resolution policies. For example, in real-time databases conflicts could be resolved based on a transaction's priority (e.g., [4], [5], [6] [7]).

C. *Virtual Execution*

Virtual execution allows those transactions that have been aborted to re-execute using in-memory values as opposed to reading directly from persistent store. Cached values from the write sets of committed transactions together with read sets from currently executing transactions populate a buffer local to the transaction management system. This can improve performance if overheads associated to persistent store access are significant. Therefore, distributed data stores [12] and real-time databases [8] have made use of such techniques.

Analysis has shown that virtual execution approaches that utilize optimistic concurrency control perform better if transactions are allowed to reach the end of their read phase before being aborted [9] [10]. This is intuitive, as transactions that have been aborted early would not have retrieved all the required data to be ready locally for the rerun phase.

D. *Contribution*

Considering the related work, our first thought was that the combination of FOCC together with virtual execution would provision an appropriate solution for our requirements. For example, the *OCC with Broadcast Rerun* described in [9] provides a reasonable starting point for an implementation. However, when considering the details of such an approach within the context of our own requirements (real-time, limited resources) the two following observations led us to consider the algorithm presented in this paper:

1. Transactions that enter rerun execute quicker than those in their initial run (as there is no persistent store access).
2. The validation phase presents a degree of non-determinism with respect to how long it will take.

As reruns are much quicker they can be achieved multiple times with minimum performance overhead. Therefore, it would be better to keep transactions in rerun until we can deterministically say that when transactions leave rerun they will complete, irrelevant of the delay imposed by the validation step. This would allow prioritization of rerun transactions without concern for non-deterministic latency in the validation phase. The novel approach advocated by our algorithm to overcome this is the contribution in this paper.

III. ALGORITHM

We now describe our algorithm by first identifying its novelty and then providing a brief overview of how the algorithm works. Pseudo code is presented to allow others to replicate our work with ease.

A. *A Different Approach*

Our algorithm fundamentally changes the order of the traditional read/validation/write phases. Write now follows

read with validation occurring after write. This is shown in figure 2.

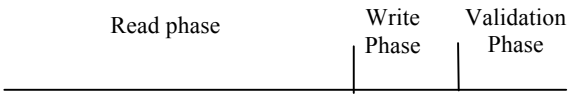

Figure 2. Algorithm phases

In addition to the reordering of the phases the algorithm makes use of a rerun policy. Those transactions that are rolled back are then rerun using in-memory data derived from retaining a buffer that records the writes of committing transactions and the reads of uncommitted transactions.

By moving the validation phase we hope to encourage the following:

- Ensure the nearest to expiring transaction (reaching their deadline) is afforded priority to commit.
- No need to block transactions during the write and validation phases to promote real-time efficiencies and allow greater determinism.

B. Description

Transactions that reach the end of the read phase join a pre-commit set. Members of the pre-commit set may be elected to proceed to the write phase by a scheduler (assuming no conflict) or be rerun (conflict). The scheduler operates a priority approach as described in [11].

Before transactions reach the write phase they may be aborted and rerun. Transactions that are in their initial run (i.e., reading directly from persistent store) are allowed to proceed to the end of the read phase and join the pre-commit set, irrelevant of conflicts. In a traditional optimistic approach without rerun such transactions may be aborted. However, our desire is to gain improved performance by retrieving persistent store data only once (as described in [9]). Therefore, only transactions in the pre-commit set, rerunning or waiting, may be aborted and rerun.

Validation is achieved by identifying the intersection between the committed transaction's write set $WS(T_c)$ and every read set of all running transactions $RS(T_i)$:

$WS(T_c) \cap RS(T_i) \neg= \phi$

If the above holds true (i.e., there is intersection), then the resolution strategy is as follows:

- If transaction is in initial run then let it proceed but mark it for rerun (to prevent it from progressing to write phase).
- If transaction is in rerun then abort. Ensure that the read sets of all aborted transactions are updated to reflect the write set of the validating transaction.

The write and validation phases are collectively considered a single critical section, only allowing one transaction at a time to enter these phases. New transactions may start the read phase at any time as:

- If a transaction starts during another transactions write phase and reads outdated persistent store then this will be captured during the later validation phase.
- If a transaction starts during another transactions validation phase then it will read updated writes from this transaction directly from the database (as validation occurs after write).

The above two properties help a great deal in that we don't delay (via blocking) transactions reading from persistent store (which is expensive). But we do retain the correctness schedule associated to the concurrency control algorithm.

Finally, after the validation phase is completed the write set associated to the transaction within the critical section is discarded.

Read-only transactions are treated slightly differently in that if they are validated as appropriate to proceed to the write phase they simply commit.

The algorithm is described thus (WS = write set; RS = read set; CS = conflicted items set; Tc = committed transaction, Ti = distinct transaction):

```
commit WS(Tc) ;
validate Ti
        foreach Ti (i=1,2,…,n) in active TS
        {
                CS := WS(Tc) ∩ RS(Ti);
                IF CS ¬= ∅  THEN
                   foreach Oj  (j=1,2,...,m) in CS
                      {read CS(Oj) from WS(Tc);}
                   update( RS(Ti), CS);
        }
discard WS(Tc);
update( RS(Ti), CS)
{
        if Ti = first run then
                wait till the end of read phase();
        foreach Oj  (j=1,2,...,m) in CS
                 {update RS(Oj)from CS;}
        rerun Ti;
}
```

The algorithm described creates schedules that are serializable (which enforces the ACID properties of databases).

If there is a complete failure of the system then on re-start all transactions that have not committed start again and read from the database directly (as if it were the first run). Any validation that was occurring during failure would be lost. However, as the transaction initiating validation has already written to the database their writes will be available on restart.

IV. Conclusions and future work

We have presented an algorithm provisioning serializability that utilizes reruns to limit repeated persistent store access of restarted transactions. We primarily do this to conserve energy on resource-constrained devices and reduce latency associated with such access. We have tailored the OCC Broadcast During Rerun as described in [9] by moving the validation phase to after the write phase. This affords more determinism in the execution as transactions can enter the read phase at any time without delaying a write due to extended validation times. Compared to research in real-time databases (e.g., [13, 14, 15]), our approach to inverting the validation and write phases is novel.

This paper provisions only the algorithm and, therefore, our initial research into concurrency control on constrained devices with real-time requirements. However, the contribution of the algorithm is significant alone due to its unique nature of allowing writes to occur before validation. In our case this allows a greater degree of determinism to be applied at the application level. This is because whichever transaction is elected to proceed from the read phase will write without delay and the write time can be known (or at least determined with some accuracy).

The algorithm does have limitations, but these reflect the context of the work (resource-constrained hand held devices). One limitation is that the transaction system must be co-located on the same device as the transactional clients to ensure the efficient management of reruns. Another limitation is that a transaction rerun must not deviate from the read/write set of previous runs.

Our future work will concentrate on validation, experimentation and implementation. We work with hand held gaming devices so we will deploy our approach on such devices to determine its effectiveness. In addition, we would like to explore how our algorithm may be applied in a distributed setting. Having the write phase before validation may seem unintuitive, but could have some interesting applications in distributed systems.

Acknowledgment

We would like to thank Fahren Bukhari for his insightful discussions regarding the development of this algorithm.

V. References

[1] Eswaran, K., P., Gray, J., N., Lorie, R., A., and Traiger, I., L. The notions of consistency and predicate locks in a database system, in Commun. ACM. p. 624-633. 1976.

[2] Kung, H., T. and Robinson, T., John, On optimistic methods for concurrency control. ACM Trans. Database Syst, 6, 2: p. 213-226. 1981.

[3] Härder†, T., Observation on Optimistic Concurrency Control Schemes. Inform. Systems, 9(2): p. 111-120, 1984.

[4] Huang, J., Stankovic, A., J., Ramamritham, K., Towsley, D., Experimental Evaluation of Real-Time Optimistic Concurrency Control Schemes, in Proc. 17th Conf. Very Large Databases. 1991. p. 35-46.

[5] Lee J., Precise serialization for optimistic concurrency control, in Elsevier Science B.V. 1999: The Netherlands. p. 163-179.

[6] Lee, J., Son, H., S., Using Dynamic Adjustment of Serialization Order for Real-time Database Systems, in 14th IEEE Real-time Systems Symposium. 1993.

[7] Lindstrom, J., Integrated and adaptive optimistic concurrency control method for real-time databases, in International Conference on Real-Time Computing Systems and Application. 2002.

[8] Franaszek, A., P., Robinson T., J., Thomasian A., Access invariance and its use in high contention environments, in Proc. Int. Conf. on Data Engineering. 1990. p. 47-55.

[9] Yu s. P., Dias, M., D., Analysis of hybrid concurrency control schemes for a high data contention environment, in IEEE Trans. Software Eng. 1992. p. 118-129.

[10] Yu s. P., Dias, M., D., Performance analysis of optimistic concurrency control schemes with different rerun policies, in Proceedings of the Fifteenth Annual International 1991: Japan p. 294 – 300

[11] Haritsa, R., J., Livny, M., Carey, J., M., Earliest deadline scheduling for real-time database systems, in Proc. 12th Real-Time System Symp. 1991.

[12] Thomasian, A., Distributed optimistic concurrency control methods for high-performance transaction processing. Knowledge and Data Engineering, IEEE Transactions on, 1998. 10(1): p. 173-189.

[13] Wang Y., W.Q., Wang H., Dai G. Dynamic adjustment of execution order in real-time database. in In Proceedings of 18th International Parallel and Distributed Processing Symposium. 2004.

[14] Lindström J., R.K., Using real-time serializability and optimitic concurrency control in firm real-time database. , in in proceedings of the 4th IEEE International Baltic Workshop on DB and IS Baltic DB IS' 2000. p. 25-37.

[15] Bai T., L.Y., Hu Y., Timestamp vector based optimistic concurrency control protocol for real-time databases, in in Wireless Communications, Networking and Mobile Computing 2008. WiCOM '08. 4th International Conference. p. 1-4.