

# Less Optimism About Optimistic Concurrency Control

C. Mohan

Data Base Technology Institute, IBM Almaden Research Center, San Jose, CA 95120, USA  
mohan@almaden.ibm.com

**Abstract** This paper attempts to document some of the shortcomings of the optimistic concurrency control (OCC) approach in supporting all the features expected in a full-function DBMS. Surprisingly, in spite of OCC having been around for a long time and its performance having been studied in various contexts, no *complete* system design, let alone a full-blown implementation, exists, as far as the author knows. The problems with OCC relate to support for access paths (indexes, hash-based storage), partial rollbacks, nested transactions, fine-granularity (e.g., record-level) conflict checking, different isolation levels, distributed transactions, and so on. The goal of this paper is to increase awareness of the implementation aspects of OCC amongst researchers and to initiate a debate about the practical utility of OCC.

## 1. Introduction

The optimistic concurrency control (OCC) approach has been around for a long time [KuRo81]. We will assume that the reader is aware of the general aspects of the OCC approach (e.g., the fact that a transaction execution consists of a *read phase* followed by a *validation phase* which is ended by a *write phase*, assuming that the validation is successful). Since the original proposal was made, many researchers have studied its performance and other characteristics [AgCL87, BrRe83, CaLi89a, CaLi89b, Care83, CeOw82, DaTK88, MeNa82, PeSM87, PrSU86, Rahm87, RaTh89, Reim83, Schl81, UnPS83, YuDi89a, YuDi89b, YuHD89]. An excellent paper written by Theo Haerder [Haer84] makes some very insightful observations and points out many of the problems with OCC. Yet, we are unaware of any serious attempts to solve these problems in the published literature. In order to increase the awareness of DB researchers about these issues, this paper fur-

ther expands on some of these problems and discusses many additional concerns with OCC.

## 2. Fine-Granularity Conflict Checking

Given that locking is the universally accepted mechanism for concurrency control and that the experience in the real-world is that fine-granularity (e.g., record) locking is crucial for good performance [GawI85, Haer88, MHLPS89, Onei86, Onei91, PeRS88, Reut82], it is unfortunate that almost all the work on OCC assumes that *page* is the smallest granularity of conflict checking. Sometimes, papers on OCC talk about *objects* being the units of conflict checking. Such papers invariably do not discuss how object creations and deletions are dealt with in the presented methods. This is a continuation of the serializability theory's (see, e.g., [BeHG87]) simplistic assumption of reads and writes being the operations to be considered, at the expense of dealing with the more interesting problems posed by operations that also include creation and deletion of data. The reader is referred to [Moha90a, Moha90c, MoLe89] for discussions concerning the latter in the context of locking.

Once it becomes essential that we deal with the fine-granularity conflict checking requirement, OCC encounters significant problems relating to storage management, access path maintenance, etc. Questions like the following would have to be dealt with:

- During inserts of records, if the storage model<sup>1</sup> used is like the one of System R [Aetal76], then when should the RIDs be assigned?

If the RID for a newly inserted record is not assigned immediately, then the access path related modifications cannot be performed until the write phase since the RID is needed for insertion with the key values in the indexes defined for the table. This means that significant number of read I/Os of index and data pages might have to be

---

<sup>1</sup> Records are stored in an area separate from pages of indexes. Each record is assigned a unique record identifier (RID). A record's RID does not change during the lifetime of the record unless a reorganization of the data takes place. Indexes contain key values and the RIDs of the records containing those key values [Moha90a, MoLe89].

performed during the write phase which would prolong that phase. This could cause a considerable reduction in concurrency if the write phases of different transactions are executed serially.

If the RID is assigned immediately, then an immediate update would have to be performed in a globally visible (to other transactions) area which will allow all transactions to know that the newly assigned RID is not available anymore for future inserts. Later, if the inserting transaction were to rollback, then somehow it must be ensured that that update is undone. This is counter to the way rollback is generally handled in OCC!

- If the updates performed during the read phase cause only private copies of the relevant data base pages to be updated, then what happens at the time of the write phase?

The write phase would have to merge the updates performed on the private versions of the data base pages with the versions of those pages as they exist in the data base at the time of the beginning of the write phase. This merging would be needed since, as a result of the fine-granularity conflict checking that is performed, other transactions might have committed in parallel with nonconflicting updates involving the same pages. Assume that the storage within a page is managed in a *logical* fashion in the interest of achieving good storage utilization in the presence of variable length records.<sup>2</sup> This would mean that conflict checking would not be based on physical locations of records, but on RIDs. Under these conditions, the merging of different versions of a page would be a nontrivial task. Sometimes, the merge may not be possible due to lack of enough free space on the page. This may necessitate rolling back the affected transaction.

- To accommodate hot spots [Gawl85, MoNS91, Onei86, Onei91, Reut82] in locking-based designs, semantically-rich lock modes are used. These permit multiple transactions' uncommitted up-

dates of a certain type (e.g., commutative update operations) to coexist even in the same field of a record [MHLPS89]. In order to support a similar notion in OCC, the conflict-checking component of OCC would have to be refined to accommodate a semantically richer set of operations on the data. The write set entries might have to be split up into many sets based on the semantics of the operations performed on the corresponding data.

- Is there an analog of hierarchical locking (lock escalation) for OCC?

If the sizes of the read and/or write sets of an executing transaction become very large, then it is conceivable to think of consolidating that information at the next higher level for the most popular tables/files (e.g., if the original granularity was page level, then after escalation pretend that the whole table was read or modified, as the case may be). The conflict checking rules would then have to take into account the multi-granularity nature of the read and write sets information.

### 3. Recovery

If the log sequence number (LSN) concept [Gray78, MHLPS89] is going to be used to ensure that at recovery time only the required redo operations are performed, then the log records describing the updates performed by a transaction cannot be written to the log file until the write phase of the transaction begins.<sup>3</sup> This is because, with the use of LSNs, the order of logging for different updates to a page must be the same as the order in which those updates were applied to the page. Of course, even without the use of LSNs, given the *no-steal* buffer management policy [HaRe83] required by OCC, we would not probably want to log anything until we are sure that a transaction is committing or at least is entering the prepared state of two-phase commit in the case of distributed transactions. The delayed logging would be required if finer than page level conflict checking is performed.

---

<sup>2</sup> It is this type of storage management that is performed in systems like System R, the OS/2 Extended Edition Data Base Manager, DB2 and so on. As a result, logging of the changes to a page is also recorded in a logical fashion. With this type of storage management, garbage collection activities which are performed to bring to a contiguous area the free space on the page need not lock or log the moved records [MHLPS89].

<sup>3</sup> Of course, the log records could still be prepared and kept aside as updates are performed during the read phase. If this is not done, then generating the log records during the write phase would be a costly activity since doing that would require comparing the private versions of the modified pages with their versions in the main data base to determine what changes were made to the pages.

Of course, if access paths are updated only during the write phase, then there is no choice with respect to when the logging can be done. This delayed logging can also prolong the duration of the write phase which could cause a reduction in concurrency if the write phases of different transactions are executed serially.

Since the no-steal buffer management policy with a private workspace is generally used in OCC designs, it may be tempting to say that, for at least nondistributed transactions, there is no need to log the undo information in the log records. This optimization has to be done carefully since it is possible that after a transaction is validated successfully, while its log records are being written, the system might fail. In this case, on restart this transaction would have to be rolled back. While there would be no uncommitted update on disk to be undone (due to the no-steal policy), for media recovery to work in a single pass, the incomplete set of log records for the transaction needs to be dealt with by writing compensation log records (CLRs) which describe the undo of those log records' updates. Even though actually no undo operations would have to be performed during restart, still the CLRs would have to be written for the sake of making media recovery work correctly. If the original log records were written without the undo information, then, in order to write the CLRs, the data base on disk must be accessed to get the undo information. More discussions concerning this problem in the context of IMS Fast Path may be found in [MHLPS89, MoTO90].

#### 4. Access Paths

When accesses are made through indexes, the *phantom* problem [EGLT76] must be taken care of.<sup>4</sup> Locking-based algorithms which address this problem are presented in [Moha90a, Moha90b, Moha90c, MoLe89]. These algorithms provide very fine-granularity locking. It is not at all obvious how these algorithms can be adapted to work with the OCC approach when fine-granularity conflict checking is necessary.

#### 5. Partial Rollbacks

In the typical OCC approach, conflicts detected during the validation phase cause the transaction to be completely rolled back. If ANSI standard SQL is to be supported by a system using OCC, then some interesting problems arise given the semantics of SQL. How is *statement-level* atomicity going to be supported? That is, if a uniqueness constraint is violated during the course of the execution of a SQL statement, then the system should undo the effects of that statement *alone* before returning control to the user. This is called **statement-level atomicity**. This requires the capability to detect violations *immediately* and also to be able to support *partial* rollback of a transaction. Under these conditions, it is not acceptable to roll back the entire transaction. Since uniqueness constraints are almost always monitored by maintaining a unique index, these requirements force index updates to be attempted in line with the processing of the user updates, rather than being delayed until the write phase of OCC. This is already in some sense counter to the major thrust of OCC which delays all updates to the main data base so that they happen during the write phase.

In System R, during a partial rollback, any locks acquired during the interval from the setting up of the savepoint which is the target of the partial rollback and the current time are released. With write-ahead logging, as explained in [MHLPS89], such locks can be released only if the same log record would not be undone more than once and compensation log records<sup>5</sup> (CLRs) are never undone. Similar actions would have to be taken with OCC also. That is, from the read and write sets, those objects that were referenced in the partially rolled back portion of the transaction must be eliminated. This requires that a time-ordered log of the additions to the read and write sets be kept and synchronized with setting up savepoints and rolling back to savepoints. Being able to perform partial rollbacks also requires that the system be logging the updates as they are performed in the private workspace and that those log records contain the undo information also.

---

4 Traditionally, for a table scan, the problem is dealt with by acquiring an S lock on the whole table.

5 CLRs are used to describe updates performed during a (partial or total) rollback.

## 6. Nested Transactions

It is not clear how OCC will support the nested transaction model. It is likely that the read and write sets of a committing child transaction would be associated with the parent, as locks are handled in locking-based system designs described in [HaRo87a, HaRo87b, Moss81, RoMo89]. But, with respect to validation, it is not obvious as to when it should be performed and which transactions the validation should be performed against. One possible approach is to perform it at the time of committing every subtransaction and do the validation against all transactions except the committing subtransaction's own ancestors, assuming that a parent transaction is blocked while the child subtransactions are executing, as was the case in Moss's model of nested transactions. It is not clear how this approach would have to be changed if parallelism is permitted between ancestors and descendants, as is the case in the very general nested transaction model of [HaRo87a, HaRo87b, RoMo89].

With respect to the private versions of the pages modified by a child subtransaction, they would have to be merged with those of the parent when the child commits. This is probably similar to the *version approach* used in [Moss81] for recovery.

## 7. Read Semantics

In the OCC approach, a transaction's updates are not installed into the main data base until the validation of the transaction's execution is successful. The updates performed by a transaction are initially applied only to the *private* versions of the pages affected by them. Under these circumstances, in the general case, it is very expensive to allow a transaction to *see* the effects of its own updates. We are aware of many applications which depend on this capability and almost all existing systems support it.<sup>6</sup> This is important, for example, in detecting integrity constraint violations like uniqueness constraints as soon as possible. It is also important in supporting active data bases with immediate triggering of actions [SPAM91].

---

<sup>6</sup> An exception is IMS Fast Path for Main Storage Data Bases (MSDBs) (see [MoTO90]). This restriction is somewhat acceptable over there since Fast Path is not a full-function DBMS, but a specialized DBMS. It has a number of restrictions on the types of operations that can be performed. Furthermore, indexes cannot be built on top of MSDBs.

## 8. Distributed Transactions

If OCC is to work for distributed transactions, then questions like the following must be addressed: How to reestablish prepared state on recovery from a failure? What gets logged and when? If the read set also needs to be logged, especially for read-only transactions, then OCC would be much more expensive than the locking approach [MoLO86].

## 9. Different Isolation Levels

Since deferred updating is employed with OCC, degree 2 consistency (also called, *cursor stability*) [GLPT76] may be easy to support since we could avoid conflict checking for data in the read set. Since all read operations read only committed data, this would provide almost the same semantics as the one provided by locking-based systems. If the read values are cached in the private area, then the reads would even be repeatable! In order to prevent other transactions from detecting unnecessary read conflicts (this is to mimic the fact that, in a locking-based system, the cursor stability transaction releases its S locks on the read data as the scan moves away from the data), the read set probably should not contain references to those objects which were only read and not modified.

## 10. Conclusion

In this paper, we attempted to briefly document the shortcomings of the optimistic concurrency control (OCC) approach in supporting all the features expected in a full-function DBMS. Surprisingly, in spite of OCC having been around for a long time and its performance having been studied in various contexts, no *complete* system design, let alone a complete implementation, exists, as far as the author knows. There have been a few prototype implementations where only a subset of the problems were addressed. We identified some of the problems with OCC. They related to supporting access paths (indexes, hash-based storage), partial rollbacks, nested transactions, fine-granularity (e.g., record-level) conflict checking, different isolation levels, distributed transactions, and so on.

## 11. References

- Aet176** Astrahan, M., et al. *System R: Relational Approach to Data Base Management*, **ACM Transactions on Database Systems**, Vol. 1, No. 2, June 1978.
- AgCL87** Agrawal, R., Carey, M., Livny, M. *Concurrency Control Performance Modeling: Alternatives and Implications*, **ACM Transactions on Database Systems**, Vol. 12, No. 4, December 1987.
- BeHG87** Bernstein, P., Hadzilacos, V., Goodman, N. *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.
- BrRe83** Braegger, R., Reimer, M. *Predicative Scheduling: Integration of Locking and Optimistic Methods*, **Research Report No. 53/**, ETH-Zurich, July 1983.
- CaLI89a** Carey, M., Livny, M. *Conflict Detection Tradeoffs for Replicated Data*, **Computer Sciences Technical Report #826**, University of Wisconsin at Madison, March 1989.
- CaLI89b** Carey, M., Livny, M. *Parallelism and Concurrency Control Performance in Distributed Database Machines*, **Computer Sciences Technical Report #831**, University of Wisconsin at Madison, March 1989.
- Care83** Carey, M. *Multiple Versions and the Performance of Optimistic Concurrency Control*, **Computer Sciences Technical Report #517**, University of Wisconsin at Madison, October 1983.
- CeOw82** Ceri, S., Owicki, S. *On the Use of Optimistic Methods for Concurrency Control in a Distributed Database*, **Proc. 8th Berkeley Workshop on Distributed Data Management and Computer Networks**, Asilomar, February 1982.
- DaTK88** Dan, A., Towsley, D., Kohler, W. *Modeling the Effects of Data and Resource Contention on the Performance of Optimistic Concurrency Control Protocols*, **Proc. 4th IEEE International Conference on Data Engineering**, Los Angeles, February 1988.
- EGLT76** Eswaran, K.P., Gray, J., Lorie, R., Traiger, I. *The Notion of Consistency and Predicate Locks in a Database System*, **Communications of the ACM**, Vol. 19, No. 11, November 1976.
- Gaw185** Gawlick, D. *Processing "Hot Spots" in High Performance Systems*, **Proc. IEEE Comcon Spring '85**, San Francisco, February 1985.
- GLPT76** Gray, J., Lorie, R., Putzolu, F., Traiger, I. *Granularity of Locks and Degrees of Consistency in a Shared Data Base*, **Proc. IFIP Working Conference on Modelling of Database Management Systems**, Freudenstadt, January 1976.
- Gray78** Gray, J. *Notes on Data Base Operating Systems*, in **Operating Systems - An Advanced Course**, R. Bayer, R. Graham, and G. Seegmuller (Eds.), **Lecture Notes in Computer Science**, Volume 60, Springer-Verlag, 1978.
- Haer84** Haerder, T. *Observations on Optimistic Concurrency Control Schemes*, **Information Systems**, Vol. 9, No. 2, 1984.
- Haer88** Haerder, T. *Handling Hot Spot Data in DB-Sharing Systems*, **Information Systems**, Vol. 13, No. 2, p155-166, 1988.
- HaRe83** Haerder, T., Reuter, A. *Principles of Transaction Oriented Database Recovery - A Taxonomy*, **Computing Surveys**, Vol. 15, No. 4, December 1983.
- HaRo87a** Haerder, T., Rothermel, K. *Concepts for Transaction Recovery in Nested Transactions*, **Proc. ACM-SIGMOD International Conference on Management of Data**, San Francisco, May 1987.
- HaRo87b** Haerder, T., Rothermel, K. *Concurrency Control Issues in Nested Transactions*, **IBM Research Report RJ5803**, IBM Almaden Research Center, August 1987.
- KuRo81** Kung, H.T., Robinson, J. *On Optimistic Methods for Concurrency Control*, **ACM Transactions on Database Systems**, Vol. 6, No. 2, p213-226, June 1981.
- MeNa82** Menasce, D., Nakanishi, T. *Optimistic Versus Pessimistic Concurrency Control Mechanisms in Database Management Systems*, **Information Systems**, Vol. 7, No. 1, 1982.
- MHLPS89** Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., Schwarz, P. *ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging*, To Appear in **ACM Transactions on Database Systems**. Also available as **IBM Research Report RJ6649**, IBM Almaden Research Center, January 1989; Revised November 1990.
- Moha90a** Mohan, C. *ARIES/KVL: A Key-Value Locking Method for Concurrency Control of Multiaction Transactions Operating on B-Tree Indexes*, **Proc. 16th International Conference on Very Large Data Bases**, Brisbane, August 1990. A different version of this paper is available as **IBM Research Report RJ7008**, IBM Almaden Research Center, September 1989.
- Moha90b** Mohan, C. *Commit\_LSN: A Novel and Simple Method for Reducing Locking and Latching in Transaction Processing Systems*, **Proc. 16th International Conference on Very Large Data Bases**, Brisbane, August 1990. Also available as **IBM Research Report RJ7344**,

- IBM Almaden Research Center, February 1990.
- Moha90c** Mohan, C. *ARIES/LHS: A Concurrency Control and Recovery Method Using Write-Ahead Logging for Linear Hashing with Separators*, IBM Research Report, IBM Almaden Research Center, March 1990.
- MoLe89** Mohan, C., Levine, F. *ARIES/IM: An Efficient and High Concurrency Index Management Method Using Write-Ahead Logging*, IBM Research Report RJ6846, IBM Almaden Research Center, August 1989.
- MoLO86** Mohan, C., Lindsay, B., Obermarck, R. *Transaction Management in the R\* Distributed Data Base Management System*, ACM Transactions on Database Systems, Vol. 11, No. 4, December 1986. Also available as IBM Research Report RJ5037, IBM Almaden Research Center, February 1986.
- MoNS91** Mohan, C., Narang, I., Silen, S. *Solutions to Hot Spot Problems in a Shared Disks Transaction Environment*, Proc. 4th International Workshop on High Performance Transaction Systems, Asilomar, September 1991.
- Moss81** Moss, J.E.B. *Nested Transactions: An Approach to Reliable Distributed Computing*, PhD Thesis, Tech Rep MIT/LCS/TR-260, MIT, April 1981.
- MoTO90** Mohan, C., Treiber, K., Obermarck, R. *Algorithms for the Management of Remote Backup Data Bases for Disaster Recovery*, IBM Research Report RJ7885, IBM Almaden Research Center, November 1990; Revised July 1991.
- Onei86** O'Neil, P. *The Escrow Transaction Method*, ACM Transactions on Database Systems, Vol. 11, No. 4, December 1986.
- Onei91** O'Neil, P. *Promises*, Proc. 4th International Workshop on High Performance Transaction Systems, Asilomar, September 1991.
- PeRS88** Peinl, P., Reuter, A., Sammer, H. *High Contention in a Stock Trading Database: A Case Study*, Proc. ACM SIGMOD International Conference on Management of Data, Chicago, June 1988.
- PeSM87** Penney, J., Stein, J., Maier, D. *Is the Disk Half Full or Half Empty?: Combining Optimistic and Pessimistic Concurrency Mechanisms in a Shared, Persistent Object Base*, Proc. Workshop on Persistent Object Systems: Their Design, Implementation and Use, Appin, August 1986.
- PrSU86** Praedel, U., Schlageter, G., Unland, R. *Redesign of Optimistic Methods: Improving Performance and Applicability*, Proc. International Conference on Data Engineering, Los Angeles, February 1986.
- Rahm87** Rahm, E. *Design of Optimistic Methods for Concurrency Control in Database Sharing Systems*, Proc. 7th International Conference on Distributed Computing Systems, Berlin, September 1987.
- RaTh89** Rahm, E., Thomasian, A. *Distributed Optimistic Concurrency Control for High Performance Transaction Processing*, Research Report RC14795, IBM T.J. Watson Research Center, July 1989.
- Reim83** Reimer, M. *Solving the Phantom Problem by Predicative Optimistic Concurrency Control*, Proc. 9th International Conference on Very Large Data Bases, Florence, October 1983.
- Reut82** Reuter, A. *Concurrency on High-Traffic Data Elements*, Proc. ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, Los Angeles, March 1982.
- RoMo89** Rothermel, K., Mohan, C. *ARIES/NT: A Recovery Method Based on Write-Ahead Logging for Nested Transactions*, Proc. 15th International Conference on Very Large Data Bases, Amsterdam, August 1989. A longer version of this paper is available as IBM Research Report RJ6650, IBM Almaden Research Center, January 1989.
- Schl81** Schlageter, G. *Optimistic Methods for Concurrency Control in Distributed Database Systems*, Proc. 7th International Conference on Very Large Data Bases, Cannes, September 1981.
- SPAM91** Schreier, U., Pirahesh, H., Agrawal, R., Mohan, C. *Alert: An Architecture for Transforming a Passive DBMS into an Active DBMS*, Proc. 17th International Conference on Very Large Data Bases, Barcelona, September 1991.
- UnPS83** Unland, R., Praedel, U., Schlageter, G. *Design Alternatives for Optimistic Concurrency Control Schemes*, Proc. 2nd International Conference on Databases, Cambridge, September 1983.
- YuDi89a** Yu, P., Dias, D. *Optimistic Concurrency Control with Adaptive Control of Abort Probability*, Research Report RC14742, IBM T.J. Watson Research Center, July 1989.
- YuDi89b** Yu, P., Dias, D. *Notes on Modelling Optimistic Concurrency Control Schemes*, Research Report RC14825, IBM T.J. Watson Research Center, August 1989.
- YuHD89** Yu, P., Heiss, H.U., Dias, D. *Optimistic Concurrency Control with Certification Based on Time-Stamp History*, Research Report RC14805, IBM T.J. Watson Research Center, July 1989.