BCC: Reducing False Aborts in Optimistic Concurrency Control with Affordable Cost for In-Memory Databases

Yuan Yuan[1], Kaibo Wang[1], Rubao Lee[1], Xiaoning Ding[2], and Xiaodong Zhang[1]

[1]The Ohio State University
[2]New Jersey Institute of Technology

Optimistic concurrency control (OCC) [8] has been implemented in the recent in-memory databases for high performance and scalability [9, 5, 13, 14, 15, 10].

With the OCC method, a database executes each transaction in three phases: read, validation and write. In the read phase, the database keeps track of what the transaction reads into a read set and buffers the transaction's writes into a write set. In the validation phase, the database validates if no change has happened to the transaction's read set. Only when the validation succeeds will the database install the transaction's writes to the database storage. Otherwise the database will abort the transaction.

OCC is optimistic in the transaction read phase. Transactions in the read phase cannot block the execution of other transactions. With the non-blocking feature, we have witnessed throughputs of over 500,000 transactions per second delivered by the in-memory OCC databases for low contention [1] OLTP workloads.

On the other hand, OCC's validation is pessimistic. To guarantee serializability, OCC will aggressively abort transactions when contention happens. However, the transaction aborts can be unnecessary. The reason is that change in a transaction's read set doesn't mean that the transaction would violate serializability requirement [7]. We call the abort of the transactions that don't affect transaction serializability *false abort*.

We use an example to illustrate OCC's false abort problem. Consider the following two transactions: T1: R(A) W(B) and T2: R(A) W(A). Figure 1 shows a schedule of T1 and T2 and the corresponding transaction Direct Serialization Graph (DSG) [1]. In this schedule, T2 can successfully commit since there are no changes in its read set. When T1 attempts to commit, the OCC database finds that data item A in T1's read set has been modified by T2. Thus T1 must be aborted according to OCC. However, since there is no cycle in the transaction DSG, T1 can actually be allowed to commit, which makes the transaction schedule equivalent to the serial execution of T1, T2. In this case, the abort of T1 is a false abort.

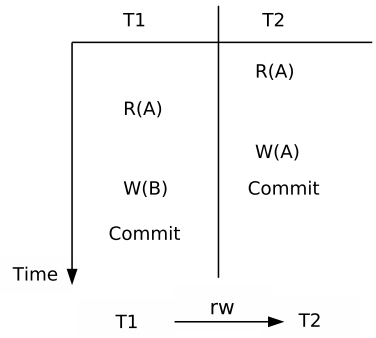

Figure 1: A schedule of T1, T2 (R:read, W:write) and the corresponding transaction DSG. T1 will be unnecessarily aborted by an OCC database.

Real world examples that have the same pattern as the above T1 and T2 include the online shopping scenario where one user is adding a new item to the shopping cart while another user has just submitted an order including the same item at the same time.

OCC's false abort problem introduces the overhead of unnecessary transaction re-execution, which is neglectable as long as transaction contentions remain low. However, with the increasing number of CPU cores and OLTP workload's inherent skew characteristics [12], transaction contention will become severe. In this case, when transaction contentions contain a large amount of read-write contentions, the overhead of false abort will significantly decrease the overall database throughput.

Essentially, as a certifier-based concurrency control method [3], OCC's false abort problem is a design tradeoff between two conflicting factors: (1) an accurate criterion to determine whether a transaction schedule can be serialized, and (2) a simple mechanism of maintaining

[1]Contention happens between transactions when one transaction's write set overlaps with the other's read set or write set [2].

transaction run-time information to help determine the criterion. OCC sacrifices the criterion's accuracy to get a potentially light-weight implementation of transaction management, which brings the benefits of performance and scalability with low contention OLTP workloads, but unavoidably causes the false abort problem.

The goal of our work is to address OCC's false abort problem without compromising OCC's benefits for low contention workloads. The idea of more precisely aborting transactions is not new. For example, [11] studied how to abort transactions based on the detection of transaction dependency cycles, similar to the deadlock detection in two-phase locking method [6]. However, these approaches introduce operations on shared data structures protected by latches for each transaction, which incur unacceptable overheads for in-memory databases [13] and are not scalable [4]. It is a challenge task to achieve a nice balance between the method's accuracy and the performance overhead.

In this work we propose a new concurrency control method called **Balanced Concurrency Control (BCC)**, which adopts the optimistic approach but has a much more accurate criterion to abort transactions. BCC's criterion is based on the detection of an essential pattern that always happens when a transaction schedule cannot be serialized. We formally prove the correctness of BCC's criterion and explain why it is more accurate than OCC's criterion.

To demonstrate BCC's effectiveness, we have implemented the BCC method in Silo [13] , which is a representative OCC-based in-memory database. We have carefully designed BCC's information bookkeeping, pattern detection approach and transaction commit protocol, and have optimized the implementations for NUMA architecture. Our implementation makes a case of how to adopt BCC in an OCC-based in-memory database.

We have comprehensively evaluated BCC's performance using both micro benchmarks and standard TPC-C benchmark on machines with up to 32 cores. Our results demonstrate that, as transaction contention increases, BCC has significant performance advantage over OCC in three aspects including reduction of transaction aborts, higher transaction throughput and better workload scalability. Meanwhile, BCC has comparable performance with OCC for low contention workloads.

References

[1] ADYA, A., LISKOV, B., AND O'NEIL, P. Generalized isolation level definitions. *2013 IEEE 29th International Conference on Data Engineering (ICDE) 0* (2000), 67.

[2] BERNSTEIN, P. A., AND GOODMAN, N. Concurrency control in distributed database systems. *ACM Comput. Surv. 13*, 2 (June 1981), 185–221.

[3] BERNSTEIN, P. A., HADZILACOS, V., AND GOODMAN, N. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[4] CLEMENTS, A. T., KAASHOEK, M. F., ZELDOVICH, N., MORRIS, R. T., AND KOHLER, E. The scalable commutativity rule: Designing scalable software for multicore processors. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP '13, ACM, pp. 1–17.

[5] DIACONU, C., FREEDMAN, C., ISMERT, E., LARSON, P.-A., MITTAL, P., STONECIPHER, R., VERMA, N., AND ZWILLING, M. Hekaton: Sql server's memory-optimized oltp engine. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2013), SIGMOD '13, ACM, pp. 1243–1254.

[6] ESWARAN, K. P., GRAY, J. N., LORIE, R. A., AND TRAIGER, I. L. The notions of consistency and predicate locks in a database system. *Commun. ACM 19*, 11 (Nov. 1976), 624–633.

[7] GRAY, J. Notes on data base operating systems. In *Operating Systems, An Advanced Course* (London, UK, UK, 1978), Springer-Verlag, pp. 393–481.

[8] KUNG, H. T., AND ROBINSON, J. T. On optimistic methods for concurrency control. *ACM Trans. Database Syst. 6*, 2 (June 1981), 213–226.

[9] LARSON, P.-A., BLANAS, S., DIACONU, C., FREEDMAN, C., PATEL, J. M., AND ZWILLING, M. High-performance concurrency control mechanisms for main-memory databases. *Proc. VLDB Endow. 5*, 4 (Dec. 2011), 298–309.

[10] NARULA, N., CUTLER, C., KOHLER, E., AND MORRIS, R. Phase reconciliation for contended in-memory transactions. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (Broomfield, CO, Oct. 2014), USENIX Association, pp. 511–524.

[11] REVILAK, S., O'NEIL, P., AND O'NEIL, E. Precisely serializable snapshot isolation (pssi). In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering* (Washington, DC, USA, 2011), ICDE '11, IEEE Computer Society, pp. 482–493.

[12] TAFT, R., MANSOUR, E., SERAFINI, M., DUGGAN, J., ELMORE, A. J., ABOULNAGA, A., PAVLO, A., AND STONEBRAKER, M. E-store: Fine-grained elastic partitioning for distributed transaction processing. *Proc. VLDB Endow. 8* (November 2014), 245–256.

[13] TU, S., ZHENG, W., KOHLER, E., LISKOV, B., AND MADDEN, S. Speedy transactions in multicore in-memory databases. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP '13, ACM, pp. 18–32.

[14] WANG, Z., QIAN, H., LI, J., AND CHEN, H. Using restricted transactional memory to build a scalable in-memory database. In *Proceedings of the Ninth European Conference on Computer Systems* (New York, NY, USA, 2014), EuroSys '14, ACM, pp. 26:1–26:15.

[15] ZHENG, W., TU, S., KOHLER, E., AND LISKOV, B. Fast databases with fast durability and recovery through multicore parallelism. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (Broomfield, CO, Oct. 2014), USENIX Association, pp. 465–477.