Efficient Deterministic Concurrency Control Under Practical Isolation Levels

Ziliang LAI
zllai@cse.cuhk.edu.hk
The Chinese University of Hong Kong

ACM Reference Format:
Ziliang LAI. 2021. Efficient Deterministic Concurrency Control Under Practical Isolation Levels. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3448016.3450580

1 INTRODUCTION

Deterministic databases are able to run transactions efficiently in a distributed setting with minimal coordination. Research has shown many benefits from having determinism, from more lightweight database replication [1, 18], to no/cheaper distributed commit [1, 20], to high throughput during live migration [11]. Besides those known applications, permissioned blockchain can also be viewed as an application of the deterministic database with security added. A permissioned blockchain maintains a fully-replicated database, where each replica is synchronized by getting blocks of updates. Early permissioned blockchains [5, 6] run a consensus protocol (e.g., PBFT [7]) to agree on the *input* and rely on the replicas to execute the transactions *serially* to uphold determinism. Fabric [3] allows concurrent transaction executions but rely on running consensus to agree on the *read-write-sets* and broadcasting them, which incurs an excessive volume of network traffic. Deterministic concurrency control combines the best of both worlds. It enables solving consensus on the small input, and each replica is able to execute transactions *independently* with high concurrency.

Although powerful, contemporary deterministic databases are still primitive because they support only one isolation level — Serializable. Furthermore, many deterministic concurrency control schemes are overly pessimistic and cause many unnecessary transaction aborts. These factors motivate us to design DCC, a suite of *better* deterministic concurrency control schemes, not only for Serializable but also for practical isolation levels *beyond* Serializable.

Motivation 1: better Serializable. Existing deterministic concurrency control schemes support only Serializable and have many issues. Early proposals such as Calvin [20], PWV [9], BOHM [8] require to know the transactions' read-write-set a priori so that the transactions can be deterministically scheduled before real execution. However, it has been shown to be impractical when the transaction logic branches based on predicates [12]. One workaround is to discover the read-write set by a trial run before the real execution, which effectively doubles latency and hurts throughput [19]. Aria [12] and BCDB [15] eliminate this requirement by *deterministic optimistic concurrency control*. Upon receiving the a batch of transactions, the replicas execute transactions against the same snapshot of the database and serializability is upheld by deterministic cycle prevention. However, to reduce the expensive cycle prevention overhead, Aria and BCDB make a trade-off and only look at the *local dependencies* around a transaction. The downside of their approach is that many transactions would be unnecessarily aborted because cycles are prevented based on incomplete information. Besides, each transaction hides its updates in thread-local copies during execution and only applies the changes upon commit. This is unfavorable for blockchains which is typically disk-based databases [3, 15], because concentrating writes during commit reduces the chance of interleaving disk I/Os with memory and CPU operations.

Motivation 2: beyond Serializable. Existing proposals are only designed for Serializable isolation level (the strongest isolation level that guarantees to process transactions in a manner that is equivalent to a serial order) [2]. Although they provide strong consistency, weaker isolation levels (e.g., Snapshot Isolation) are prevalent in practice and overwhelmingly the default setting in commercial databases [4], because they offer much higher concurrency while allow the application level to handle consistency issues only when necessary [2] (e.g., TPC-C [17] runs safely under Snapshot Isolation without special care in application level [10]). To fill the gap, we design the first set of deterministic concurrency control algorithms DCC-RC, DCC-RR and DCC-SI for Read Committed, Repeatable Read and Snapshot Isolation, respectively, in addition to DCC-SE for Serializable. Furthermore, DCC allows mixing these isolation levels such that an application can designate an ad hoc isolation level for each transaction. Although there exists other isolation levels (e.g., Cursor Stability [2]), we choose this set because they are most commonly seen in commercial databases [4].

Besides, permissioned blockchains are also showing the trend of relaxing consistency for better performance. For instance, HyperLedger Fabric [3] lets applications to handle phantom reads (if necessary) for better performance in its *private data collections* [14]. As in relational databases, Serializable is also overkill for many blockchain applications. For example, a common blockchain application that handle payment transactions (e.g., A pays B $10 if A has enough balance) work correctly and more efficiently under Snapshot Isolation. As far as we know, we are the first to introduce the notion of weaker isolation levels in blockchain world, and the value of doing so was acknowledged in our contact with a IBM blockchain team.

SIGMOD '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8343-1/21/06.
https://doi.org/10.1145/3448016.3450580

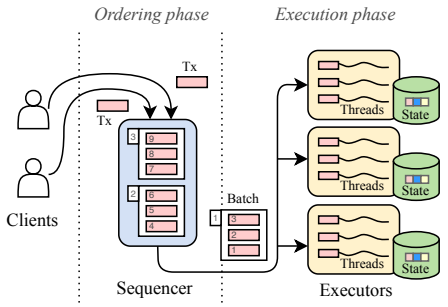

Figure 1: The order-then-execute workflow

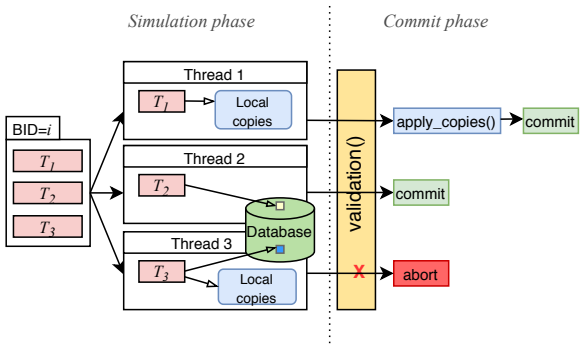

Figure 2: The transaction flow of DCC

2 BACKGROUND

Deterministic databases mostly follow an *order-then-execute* workflow [8, 9, 12, 20] illustrated in Figure 1. In ordering phase, transactions are sent to a sequencer where they are assigned an unique ID (`TID`) and packed into batches. The batches are attached to an incremental sequence number (`BID`) and then broadcast to executors (replicas) who carryout deterministic concurrency control to process transactions *batch by batch*. In this workflow, the sequencer does no heavy-lifting but ensures that the replicas receive identical batches of transactions. In addition, it hard-codes non-deterministic operations (e.g., `RAND` and `TIME`) with constants so that they do not cause non-deterministic behaviors [1]. The sequencer can reside on a single machine, or can be jointly maintained by a cluster for fault-tolerance and scale-out [19]. Interestingly, blockchain systems often exhibit the same workflow [6, 13, 21], where a cluster of participants runs a consensus protocol (e.g., PBFT [7]) to form the sequencer, and other (or perhaps the same set of) participants act as replicas.

3 APPROACH

Aiming for practical use, DCC adopts deterministic optimistic concurrency control approach to avoid the requirement of predetermining read-write-set. But unlike Aria and BCDB, DCC aims to support practical isolation levels, eliminate unnecessary aborts and unnecessary thread-local copies. For Serializable, DCC overcomes the challenge of expensive cycle prevention by uncovering new properties about transaction dependencies so that it can examine a *minimal but sufficient* view of the dependency graph to *totally avoid unnecessary aborts with almost no overhead*. For weaker isolation levels, instead of simply relaxing its Serializable implementation, DCC maximizes the performance of each isolation level by uncovering some isolation-level specific properties. Furthermore, DCC supports *isolation mixing* [2], i.e., the use of different isolation levels for different transactions in a workload. Mixing isolation level is challenging because mixing the individual optimizations of each isolation level may undermine the overall correctness [2]. Besides, DCC is designed to make thread-local copies only when conflicting updates are found.

Transaction flow. DCC acts as the core of the executor in Figure 2. Upon receiving a batch of transactions, DCC processes it by a simulation phase and a commit phase as illustrated in Figure 2. In the simulation phase, DCC executes transactions concurrently against the snapshot that captures the final state of the previous batch. Unlike Aria and BCDB, an update/delete operation is applied to the database directly if no other transactions in the current batch has updated/deleted the same record (e.g., T_1). Otherwise, the update/delete is stored as a thread-local copy (e.g., T_2). Insertions are also performed directly if no unique-key constraint is violated. In this way, DCC saves the cost of making thread-local copies, which is particularly favorable for low-conflict workloads because most of the transactions can directly apply changes to the database. After all the transactions in current batch finish simulation, DCC enters commit phase where it invokes `validation()` to deterministcally abort transactions that violates the consistency requirements of the specified isolation level. For each transaction that passes validation, if thread-local copies exist, it invokes `apply_copies()` to resolve the conflicting updates based on the semantic of the isolation level and applies changes to the database. Otherwise, the transaction can directly commit. `validation()` and `apply_copies()` can be instantiated to support various isolation levels.

Cycle prevention. Historically, cycle prevention incurs high overhead as it has to trace long chains of dependencies and it must be invoked every time a transaction intends to commit [16]. We overcome its overhead by several observations: 1) Dependency cycles do not span across different batches; 2) wr-dependencies do not cause cycles; 3) ww-dependencies coexist with rw-dependency circles; With these observations, our deterministic cycle prevention algorithm only needs to examine a minimum dependency graph containing *only transactions in current batch* and *only rw-dependencies*.

4 CONTRIBUTION

To summarize, the paper makes the following principle contributions. 1) We provide the first set of deterministic concurrency control algorithms for practical isolation levels including Read Committed, Snapshot Isolation, Repeatable Read besides Serializable. Our proposed algorithm, DCC also supports mixing these isolation levels in one application. 2) Even for Serializable, we propose novel approaches to avoid unnecessary aborts and unnecessary thread-local copies and thereby allow more concurrency and offer higher throughput. With the optimizations discussed above, DCC-SE is expected to achieves 2-3 times higher throughput than BCDB and Aria. DCC-RC, DCC-RR and DCC-SI must gain even larger margin.

REFERENCES

[1] Daniel J Abadi and Jose M Faleiro. 2018. An overview of deterministic database systems. *Commun. ACM* 61, 9 (2018), 78–88.

[2] Atul Adya and Barbara H Liskov. 1999. *Weak consistency: a generalized theory and optimistic implementations for distributed transactions.* Ph.D. Dissertation. Massachusetts Institute of Technology, Dept. of Electrical Engineering and ....

[3] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*. 1–15.

[4] Peter Bailis. [n.d.]. Highly Available, Seldom Consistent. http://www.bailis.org/blog/when-is-acid-acid-rarely/

[5] Arati Baliga, I Subhod, Pandurang Kamat, and Siddhartha Chatterjee. 2018. Performance evaluation of the quorum blockchain platform. *arXiv preprint arXiv:1809.03421* (2018).

[6] Ethan Buchman. 2016. *Tendermint: Byzantine fault tolerance in the age of blockchains.* Ph.D. Dissertation.

[7] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.

[8] Jose M Faleiro and Daniel J Abadi. 2014. Rethinking serializable multiversion concurrency control. *arXiv preprint arXiv:1412.2324* (2014).

[9] Jose M Faleiro, Daniel J Abadi, and Joseph M Hellerstein. 2017. High performance transactions via early write visibility. *Proceedings of the VLDB Endowment* 10, 5 (2017).

[10] Alan Fekete. 2009. *Snapshot Isolation.* Springer US, Boston, MA, 2659–2664. https://doi.org/10.1007/978-0-387-39940-9_346

[11] Yu-Shan Lin, Shao-Kan Pi, Meng-Kai Liao, Ching Tsai, Aaron Elmore, and Shan-Hung Wu. 2019. MgCrab: transaction crabbing for live migration in deterministic database systems. *Proceedings of the VLDB Endowment* 12, 5 (2019), 597–610.

[12] Yi Lu, Xiangyao Yu, Lei Cao, and Samuel Madden. 2020. Aria: a fast and practical deterministic OLTP database. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2047–2060.

[13] Satoshi Nakamoto. 2019. *Bitcoin: A peer-to-peer electronic cash system.* Technical Report. Manubot.

[14] Senthil Nathan. [n.d.]. http://www.bchainledger.com/2020/06/performance-analysis-of-private-data.html

[15] Senthil Nathan, Chander Govindarajan, Adarsh Saraf, Manish Sethi, and Praveen Jayachandran. 2019. Blockchain meets database: Design and implementation of a blockchain relational database. *arXiv preprint arXiv:1903.01919* (2019).

[16] Dan RK Ports and Kevin Grittner. 2012. Serializable snapshot isolation in PostgreSQL. *arXiv preprint arXiv:1208.4179* (2012).

[17] Francois Raab. 1993. TPC-C-The Standard Benchmark for Online transaction Processing (OLTP).

[18] Kun Ren, Dennis Li, and Daniel J Abadi. 2019. Slog: Serializable, low-latency, geo-replicated transactions. *Proceedings of the VLDB Endowment* 12, 11 (2019).

[19] Kun Ren, Alexander Thomson, and Daniel J Abadi. 2014. An evaluation of the advantages and disadvantages of deterministic database systems. *Proceedings of the VLDB Endowment* 7, 10 (2014), 821–832.

[20] Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, and Daniel J Abadi. 2012. Calvin: fast distributed transactions for partitioned database systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 1–12.

[21] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.