Efficiently Supporting Adaptive Multi-Level Serializability Models in Distributed Database Systems

Zhanhao Zhao[*]
Renmin Univeristy of China
zhanhaozhao@ruc.edu.cn

ABSTRACT

Informally, serializability means that transactions appear to have occurred in some total order. In this paper, we show that only the serializability guarantee with some total order is not enough for many real applications. As a complement, extra partial orders of transactions, like real-time order and program order, need to be introduced. Motivated by this observation, we present a framework that models serializable transactions by adding extra partial orders, namely *multi-level serializability models*. Following this framework, we propose a novel concurrency control algorithm, called bi-directionally timestamp adjustment (BDTA), to supporting multi-level serializability models in distributed database systems. We integrate the framework and BDTA into Greenplum and Deneva to show the benefits of our work. Our experiments show the performance gaps among serializability levels and confirm BDTA achieves up to 1.7× better than state-of-the-art concurrency control algorithms.

ACM Reference Format:
Zhanhao Zhao. 2021. Efficiently Supporting Adaptive Multi-Level Serializability Models in Distributed Database Systems. 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/10.1145/3448016.3450579

1 INTRODUCTION

Thus far, almost every major RDBMS is *multi-version concurrency control* based to achieve serializability. However, recent studies engendered by decentralized database systems under multi-version concurrency control show that serializability guarantee could still lead to *stale reads*. Take transactions shown in the top part of Figure 1 as an example. Transaction T_3 starts after T_1 commits, and T_1 creates a new version x_1 of tuple x. A stale read happens since the read of T_3 returns x_0 but not x_1. This is possible, due to a lack of a globally synchronized clock, for a database system to execute T_3 over a state of the database before T_1 exists (e.g., T_1's write cannot be seen by T_3 since the snapshot of T_3 is less than the commit timestamp of T_1). Strict serializability, in essence, adds a simple additional constraint, i.e., linearizable consistency [15], which preserves the real-time order, $\prec$, of non-concurrent operations (i.e., $op_1 \prec op_2$ if operation op_2 starts after operation op_1 completes), on top of serializability to prevent stale reads. That is, strict serializability guarantees the properties of both serializability and linearizable consistency hold during the entire execution of transactions.

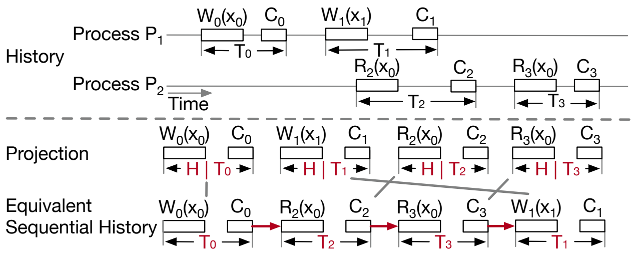


Figure 1: Formulation of Multi-level Serializability

Except Spanner [2, 9], the majority of NewSQL systems, including CockroachDB [25], TiDB [16] and MongoDB [27], do not support strict serializability. The reason behind this is not the difficulty of the implementation, but the costly overhead to request globally ordered timestamps to preserve real-time order. Consider that linearizable consistency is the strongest consistency model in distributed systems, while a weaker consistency model yields higher performance. We are therefore motivated to provide a framework to optimize between the performance and consistency in distributed database systems based on the requirements of applications.

In this paper, we aim to address the issue of serializability with multi-level consistency models, which include linearizable consistency (LC), sequential consistency (SC) [18], and causal consistency (CC) [17], etc. Based on the observation that the consistency model is defined by *selectively preserving certain kind of partial orders among operations* and serializability is defined by *preserving certain kind of total orders among transactions*, We first propose a framework by re-defining the consistency models in terms of the transaction granularity rather than the operation granularity. We theoretically analyze that all possible combinations are limited to serializability & LC (namely *strict serializability*), and serializability & SC (namely *sequential serializability*), while the other combinations are reduced to sequential serializability.

Based on the framework, we propose a unified concurrency control algorithm with various optimizations. We attach each transaction with a timestamp interval (denoted as [LB, UB]), and propose the bi-directionally timestamp adjustment (BDTA) algorithm to dynamically tracking the order of transactions by comparing and coordinating timestamp intervals of transactions. We then integrate the framework and BDTA into Greenplum [13] to study the system performance under multi-level serializability models. Besides, we implement BDTA and state-of-the-art concurrency control algorithms in Deneva [14] to compare BDTA with other algorithms. The results show BDTA achieves up to 1.7× performance gain.

[*]I would like to thank Wei Lu, Haixiang Li and Xiaoyong Du for advising this work and the following grants: National Natural Science Foundation of China 61972403, and Tencent Rhino-Bird Joint Research Program.

2 FRAMEWORK

We propose a framework to model serializability transactions by integrating consistency models in distributed systems. Most existing works [1, 10, 28, 29] focus on the combinations between a weak isolation level [4] and a weak consistency model [7]. A few works discuss strict serializability and serializability [3, 23, 24]. Differing from existing works, we make a systematic analysis of possible combinations between serializability with consistency models. We define multi-level serializability below:

Sequential History. A history H is sequential if for any two transactions T_i and T_j in H, either the last event of $H|T_i$ precedes the first event of $H|T_j$ or the last event of $H|T_j$ precedes the first event of $H|T_i$.[*] We denote $T_i \longrightarrow T_j$ if T_i precedes T_j in H.

Write Legal. A sequential history H is *write legal* if for any $H|x$ [†], any read of x_m (mth version of x) must immediately follow the write of x_m when excluding all other read events of x in $H|x$.

Partial Order. We extend the partial orders introduced by consistency models from the operation granularity to transaction granularity. Formally, given two transactions T_i and T_j in H, Program order $<_H^{pr}$: $T_i <_H^{pr} T_j$ if transactions T_i and T_j are in the same process P and the last event of T_i precedes the first event of T_j; Write-read order $<_H^{wr}$: $T_i <_H^{wr} T_j$ if there exists an operation op_1 in T_i and another operation op_2 in T_j such that op_1 reads a version written by op_2; Casual-related order $<_H^{cr}$: $T_i <_H^{cr} T_j$ if (a) $T_i <_H^{pr} T_j$ or (b) $T_i <_H^{wr} T_j$, or they are related by a transitive closure leveraging (a) and/or (b); Real-time order $<_H^{rt}$: $T_i <_H^{rt} T_j$ if the last event of T_i precedes the first event of T_j.

Multi-level Serializability Modeling. As shown in Figure 1, given a history H, we check whether an equivalent history [‡] to H can be sequential, write-legal, and can preserve the partial orders required by a specific serializability level. Following these rules, we propose the framework by defining strict serializability (SER-L), sequential serializability (SER-S), and serializability (SER) levels below. Given a history H, H satisfies

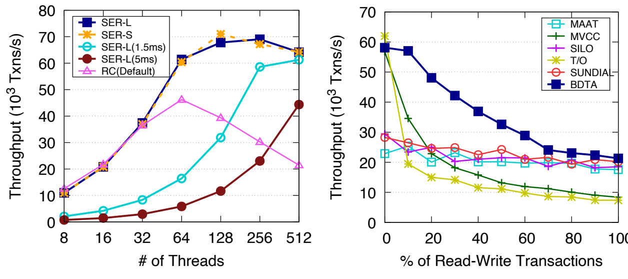
- **SER-L level** if H is equivalent to some sequential history S_H, which is write legal, with $<_H^{cr} \subseteq <_{S_H}^{cr}$ and $<_H^{rt} \subseteq <_{S_H}^{rt}$.
- **SER-S level** if H is equivalent to some sequential history S_H, which is write legal, with $<_H^{cr} \subseteq <_{S_H}^{cr}$.
- **SER level** if there is a sequential history S_H, with $S_H|T = H|T$ for each T in H, and S_H is write legal, with $<_H^{wr} \subseteq <_{S_H}^{wr}$.

3 CONCURRENCY CONTROL PROTOCOL

Our BDTA algorithm is multi-version based, and follows the optimistic way enhanced with bi-directionally dynamic timestamp adjustment to do concurrency control.

The main idea of BDTA is described as follows. Firstly, a transaction T should preserve partial orders required by the specific serializability level when T starts. In our algorithm, this property is guaranteed by snapshot isolation. We generate a snapshot from either Timestamp Oracle [22] or HLC [11] to preserve the partial order required by SER-L or SER-S, respectively. Secondly, a transaction T guarantees a total order for timestamp intervals with all concurrent transactions [§] when T commits, i.e., for any two concurrent transactions T_i, T_j, we can have either $T_i.UB < T_j.LB$ or $T_j.UB < T_i.LB$, leading to an order of $T_i \longrightarrow T_j$, or $T_j \longrightarrow T_i$, respectively. A transaction T aborts if no legal timestamp interval exists, i.e., $T.UB < T.LB$. We shall give an example shown in Figure 1 to elaborate on the BDTA mechanism. Focused on transaction T_1, when T_1 enters the validation phase, it detects T_2 has read tuple x with version x_0, and hence T_1 actively adjusts its timestamp interval with T_2 to let $T_2.UB < T_1.LB$, indicating that T_2 is ordered before T_1. This adjustment is *bi-directional*, which refines timestamp intervals of both T_1 and T_2, to preserve the partial order $T_2 \longrightarrow T_1$, causing $T_2.UB < T_1.LB$. After T_1 passing the validation phase, the commit timestamp of T_1 and x_1 will be assigned with $T_1.LB$.

BDTA algorithm brings the following benefits. First, validations for read-only transactions are eliminated, which bring performance gain in read-intensive workload compared with single-version based [20, 30, 31] and other timestamp interval based [19] algorithms. We use timestamp intervals to maintain the order of transactions, which significantly reduces the network transferring overhead against dependency graph based algorithms that shuffle read-/write set instead [12, 21]. Further, databases equipped with BDTA can uniformly support SER-L, SER-S, and SER level, which is capable of running at an appropriate level according to the requirements of applications.



(a) Multi-level Serializability (b) Concurrency Control Evaluations

Figure 2: Verification on Greenplum and Deneva

4 VERIFICATION

As shown in Figure 2(a), we first verify the performance of multi-level serializability models. We use the workload-a in YCSB [8] and vary the network latency when communicating with Timestamp Oracle. There is an obvious performance gap among different networks between SER-L (when RTT is set to $1.5ms$ or $5ms$) and SER-S since SER-S does not suffer from the increasing network latency. Besides, BDTA outperforms the default lock-based concurrency control algorithm which provide the read commited in Greenplum.

In Figure 2(b), we then confirm that BDTA performs up to 1.7× better than the next-best concurrency control algorithm. We use the same settings provided in [14] and vary the percentage of read-write transactions. Benefits with the transaction reordering ability, transactions should be aborted in the other algorithms [5, 6, 20, 26, 31], can be successfully committed in some cases by BDTA.

[*] A history H is a finite sequence of operation events generated by transactions. We assume each process executes transactions sequentially. $H|T_i$ represents the projection of T_i in H, i.e., a sequence of all events in H whose operations are from T_i.

[†] $H|x$ denotes a subsequence of all events in H whose operations executed on item x.

[‡] H and H' are said to be equivalent if for every process P, $H|P = H'|P$. $H|P$ denotes a subsequence of all events in H whose process names are P.

[§] A transaction T' is said to be concurrent to T if T' starts before T commits and commits after T starts.

REFERENCES

[1] Atul Adya. 1999. Weak consistency: a generalized theory and optimistic implementations for distributed transactions. (1999).

[2] David F. Bacon, Nathan Bales, Nicolas Bruno, Brian F. Cooper, Adam Dickinson, Andrew Fikes, Campbell Fraser, Andrey Gubarev, Milind Joshi, Eugene Kogan, Alexander Lloyd, Sergey Melnik, Rajesh Rao, David Shue, Christopher Taylor, Marcel van der Holst, and Dale Woodford. 2017. Spanner: Becoming a SQL System. In *SIGMOD Conference*. ACM, 331–343.

[3] Peter Bailis, Aaron Davidson, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. 2013. Highly Available Transactions: Virtues and Limitations. *PVLDB* 7, 3 (2013), 181–192.

[4] Hal Berenson, Philip A. Bernstein, Jim Gray, Jim Melton, Elizabeth J. O'Neil, and Patrick E. O'Neil. 1995. A Critique of ANSI SQL Isolation Levels. In *SIGMOD Conference*. ACM Press, 1–10.

[5] Philip A. Bernstein and Nathan Goodman. 1981. Concurrency Control in Distributed Database Systems. *ACM Comput. Surv.* 13, 2 (1981), 185–221.

[6] Philip A. Bernstein and Nathan Goodman. 1983. Multiversion Concurrency Control - Theory and Algorithms. *ACM Trans. Database Syst.* 8, 4 (1983), 465–483.

[7] Sebastian Burckhardt, Alexey Gotsman, and Hongseok Yang. 2013. Understanding eventual consistency. *Microsoft Research Technical Report MSR-TR-2013-39* (2013).

[8] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *SoCC*. ACM, 143–154.

[9] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson C. Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2012. Spanner: Google's Globally-Distributed Database. In *OSDI*. USENIX Association, 251–264.

[10] Natacha Crooks, Youer Pu, Lorenzo Alvisi, and Allen Clement. 2017. Seeing is Believing: A Client-Centric Specification of Database Isolation. In *PODC*. ACM, 73–82.

[11] Murat Demirbas, Marcelo Leone, Bharadwaj Avva, Deepak Madeppa, and Sandeep Kulkarni. 2014. Logical physical clocks and consistent snapshots in globally distributed databases. (2014).

[12] Bailu Ding, Lucja Kot, and Johannes Gehrke. 2018. Improving Optimistic Concurrency Control Through Transaction Batching and Operation Reordering. *PVLDB* 12, 2 (2018), 169–182.

[13] Greenplum. https://greenplum.org/.

[14] Rachael Harding, Dana Van Aken, Andrew Pavlo, and Michael Stonebraker. 2017. An Evaluation of Distributed Concurrency Control. *PVLDB* 10, 5 (2017), 553–564.

[15] Maurice Herlihy and Jeannette M. Wing. 1990. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Trans. Program. Lang. Syst.* 12, 3 (1990), 463–492.

[16] Dongxu Huang, Qi Liu, Qiu Cui, Zhuhe Fang, Xiaoyu Ma, Fei Xu, Li Shen, Liu Tang, Yuxing Zhou, Menglong Huang, Wan Wei, Cong Liu, Jian Zhang, Jianjun Li, Xuelian Wu, Lingyu Song, Ruoxi Sun, Shuaipeng Yu, Lei Zhao, Nicholas Cameron, Liquan Pei, and Xin Tang. 2020. TiDB: A Raft-based HTAP Database. *Proc. VLDB Endow.* 13, 12 (2020), 3072–3084.

[17] Leslie Lamport. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM* 21, 7 (1978), 558–565.

[18] Leslie Lamport. 1979. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. *IEEE Trans. Computers* 28, 9 (1979), 690–691.

[19] David B. Lomet, Alan Fekete, Rui Wang, and Peter Ward. 2012. Multi-version Concurrency via Timestamp Range Conflict Management. In *ICDE*. IEEE Computer Society, 714–725.

[20] Hatem A. Mahmoud, Vaibhav Arora, Faisal Nawab, Divyakant Agrawal, and Amr El Abbadi. 2014. MaaT: Effective and scalable coordination of distributed transactions in the cloud. *PVLDB* 7, 5 (2014), 329–340.

[21] Shuai Mu, Yang Cui, Yang Zhang, Wyatt Lloyd, and Jinyang Li. 2014. Extracting More Concurrency from Distributed Transactions. In *OSDI*. USENIX Association, 479–494.

[22] Daniel Peng and Frank Dabek. 2010. Large-scale Incremental Processing Using Distributed Transactions and Notifications. In *OSDI*. USENIX Association, 251–264.

[23] Kun Ren, Dennis Li, and Daniel J. Abadi. 2019. SLOG: Serializable, Low-latency, Geo-replicated Transactions. *Proc. VLDB Endow.* 12, 11 (2019), 1747–1761.

[24] Adriana Szekeres and Irene Zhang. 2018. Making consistency more consistent: a unified model for coherence, consistency and isolation. In *PaPoC@EuroSys*. ACM, 7:1–7:8.

[25] Rebecca Taft, Irfan Sharif, Andrei Matei, Nathan VanBenschoten, Jordan Lewis, Tobias Grieger, Kai Niemi, Andy Woods, Anne Birzin, Raphael Poss, Paul Bardea, Amruta Ranade, Ben Darnell, Bram Gruneir, Justin Jaffray, Lucy Zhang, and Peter Mattis. 2020. CockroachDB: The Resilient Geo-Distributed SQL Database. In *SIGMOD Conference*. ACM, 1493–1509.

[26] Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden. 2013. Speedy transactions in multicore in-memory databases. In *SOSP*. ACM, 18–32.

[27] Misha Tyulenev, Andy Schwerin, Asya Kamsky, Randolph Tan, Alyson Cabral, and Jack Mulrow. 2019. Implementation of Cluster-wide Logical Clock and Causal Consistency in MongoDB. In *SIGMOD Conference*. ACM, 636–650.

[28] Paolo Viotti and Marko Vukolic. 2016. Consistency in Non-Transactional Distributed Storage Systems. *ACM Comput. Surv.* 49, 1 (2016), 19:1–19:34.

[29] Chao Xie, Chunzhi Su, Manos Kapritsos, Yang Wang, Navid Yaghmazadeh, Lorenzo Alvisi, and Prince Mahajan. 2014. Salt: Combining ACID and BASE in a Distributed Database. In *OSDI*. USENIX Association, 495–509.

[30] Xiangyao Yu, Andrew Pavlo, Daniel Sánchez, and Srinivas Devadas. 2016. TicToc: Time Traveling Optimistic Concurrency Control. In *SIGMOD Conference*. ACM, 1629–1642.

[31] Xiangyao Yu, Yu Xia, Andrew Pavlo, Daniel Sánchez, Larry Rudolph, and Srinivas Devadas. 2018. Sundial: Harmonizing Concurrency Control and Caching in a Distributed OLTP Database Management System. *PVLDB* 11, 10 (2018), 1289–1302.