CAUSALDOT: Causally Consistent Transactions with Non-blocking Reads and Deterministic Ordering

(Relevant In-progress Research Paper)

Ruijie Gong, Haoze Song, and Heming Cui
The University of Hong Kong
{rjgong,hzsong,heming}@cs.hku.hk

Modern decentralized applications call for a new design for storage backend. In this paper, we introduce CAUSALDOT, the first deterministic database that guarantees causal consistency. CAUSALDOT is designed explicitly for edge-oriented decentralized applications. Traditional deterministic databases, such as Calvin and Detock, ensure strict serializability, which, while providing strong guarantees, introduces significant performance bottlenecks in geo-distributed edge environments. CAUSALDOT addresses these limitations by adopting a causal consistency model, which is sufficient for most edge applications and reduces the overhead associated with cross-regional communication. By leveraging deterministic transaction execution and non-blocking reads, CAUSALDOT ensure efficient and consistent transaction processing without needing a centralized ordering service or complex dependency graphs. Our approach significantly reduces transaction latency and enhances the performance of single-region transactions, which constitute the majority of operations in edge databases. As an in-progress research paper, we detail the protocol design and show the correctness of CAUSALDOT, demonstrating its potential to improve the scalability and reliability of edge databases while providing a balanced trade-off between availability and consistency.

I. INTRODUCTION

Nowadays, edge-oriented decentralized applications are becoming increasingly popular, and edge databases have been a new backend design paradigm for these data-intensive applications [1] [2] [3]. Due to these applications' demand for high scalability and the nature of cross-regional deployment (i.e., the devices and networks are geo-distributed [4]), the design and implementation of edge databases face numerous challenges.

Fortunately, deterministic databases, which have attracted much attention in recent years, can effectively mitigate the impact of network and device instability in edge environments. A deterministic database ensures that as long as all (participating) nodes reach a consensus on the input, the execution result will be the same across all these (participating) nodes. The key to deterministic databases is eliminating non-deterministic program logic [5] and letting nodes execute independently. Unlike traditional non-deterministic databases, deterministic databases integrate transaction concurrency control and replication layer protocols, ensuring that non-deterministic failures at a single node (e.g., memory or disk failure) will not abort transactions [5].

Specifically, in traditional databases, transactions are first submitted through the concurrency control layer, then the databases leverage a replication layer protocol (e.g., Paxos [6]) to achieve high reliability. Single-point failures in the concurrency control layer can cause transactions to abort and roll back, which can be common in edge scenarios. Conversely, in deterministic databases, transactions can be executed normally on other peer nodes by merging the concurrency control and replication layers, even if a failure occurs on a specific node. The failure node can recover later by deterministically replaying the input from transaction logs. As a result, unifying the two layers significantly reduces the likelihood of aborts, which are caused by device instability in edge scenarios.

Moreover, the deterministic approach can significantly enhance the performance of edge databases. This is because, as mentioned previously, edge databases are commonly distributed across regions, and the overhead of cross-regional communication is extremely high (i.e., reaching hundreds of milliseconds [7]). The deterministic model enables transactions to reach consensus with minimal cross-regional communication round trips, greatly reducing transaction latency.

However, simply deploying existing deterministic databases in edge scenarios incurs new performance bottlenecks. To our knowledge, current state-of-the-art deterministic databases have two fundamental design templates. The first, represented by Calvin [8] and SLOG [9], uses a centralized ordering service to achieve a serial order, requiring globally ordering cross-regional transactions. When an edge database involves massive regions [10], this global coordination service can easily become a performance bottleneck.

The second design, represented by Detock [11], uses a global dependency graph for cross-regional transactions to ensure deterministic execution instead of a centralized ordering service. However, like Calvin and SLOG, single-region transactions are still required to be executed after their dependent cross-regional transactions are completed. Single-region transactions may be significantly delayed when cross-regional transactions form a complex dependency graph. Since the majority of transactions in edge databases (90% [12]) are single-region transactions, this can lead to severe latency for single-region transactions.

Our insight is that the strongest consistency level provided

by SLOG and Detock becomes a key factor limiting their performance. In particular, SLOG and Detock implement strict serializability, which provides the strongest guarantees. However, we found such a consistency model is too heavy and expensive. As a neutral point in the trade-off between availability and consistency, causal consistency is sufficient to meet the requirement of most edge applications (e.g., web-browsing caches [13]), guaranteeing that an update does not become visible until all its causal dependencies are visible [14]. For example, imagine Alice uploading a new photo to her social media profile, and shortly afterward, Bob, who follows Alice, comments on the photo. In a system that maintains causal consistency, any user who sees Bob's comment must also be able to see Alice's photo.

Therefore, we propose CAUSALDOT, the first deterministic database model that guarantees causal consistency. CAUSALDOT is based on Detock, which is currently the most performant deterministic algorithm to date. While retaining the advantage of Detock's non-requirement for single-node sorting, CAUSALDOT ensure non-blocking execution of single-region transactions and avoids the delay in constructing a complete cross-regional dependency graph, meeting the requirements of edge transactions. At the same time, CAUSALDOT also provides more potential possibilities for future deterministic database design in terms of consistency choice.

Roadmap. The remainder of the paper is organized as follows. Section III presents the background. Section IV describes the protocols and correctness of CAUSALDOT. Section V discusses related work. Section VI concludes the paper.

II. BACKGROUND

A. Causal Consistency

A system is causally consistent if its servers return values that are consistent with the order defined by the *causality* relationship. Causality is defined as a happens-before relationship between two events [15], [16]. For two operations a, b, we say that b causally depends on a, and write $a \rightsquigarrow b$, if and only if at least one of the following conditions holds: $i)$ a and b are operations in a single thread of execution, and a happens before b; $ii)$ a is a write operation, b is a read operation, and b reads the version written by a; $iii)$ there is some other operation c such that $a \rightsquigarrow c$ and $c \rightsquigarrow b$. Intuitively, CC ensures that if a client has seen the effects of operation b and $a \rightsquigarrow b$, then the client also sees the effects of operation a.

We use lower case letters, e.g., x, to refer to a key and the corresponding capital letter, e.g., X to refer to a version of the key. We say that X depends on Y if the write of X causally depends on the write of Y.

B. Transactional Causal Consistency

TCC extends CC by means of interactive read-write transactions in which clients can read and write multiple items. TCC enforces two properties.

1. Read from a causal snapshot. A causally consistent snapshot is a set of item versions such that all causal dependencies of those versions are also included in the snapshot. All transactions read from a causally consistent snapshot. For any two items, x and y, if $X \rightsquigarrow Y$ and both X and Y belong to the same causally consistent snapshot, then there is no other X', such that X' is created after X and $X \rightsquigarrow X' \rightsquigarrow Y$.

Transactional reads from a causal snapshot prevent undesirable anomalies which can arise by simply issuing multiple consecutive single read operations [17].

The majority of existing CC systems implement transactional reads by means of one-shot read-only transactions [17]–[20].

2. Atomic updates. Either all the items written by a transaction are visible to other transactions, or none is. If a transaction writes X and Y, then any snapshot visible to other transactions either includes both X and Y, or none of them.

III. SYSTEM ARCHITECTURE

CAUSALDOT is a geo-partitioned database system, deployed across multiple geographic regions. A region consists of servers connected via a low-latency network. These servers typically reside within a single data-center or multiple data-centers that are in close proximity with each other. Each item is assigned to exactly one geographic region. This is called a home region in both SLOG and Detock.

A region may store local data for which it is designated as the home region, and remote data which are materialized by replaying logs asynchronously replicated from other regions. Data is also partitioned locally within a region independently of home status: each partition might contain a mix of local and remote data.

In addition, each region is equipped with a physical clock. Clocks are loosely synchronized by a time synchronization protocol such as NTP [21]. Each clock generates monotonically increasing timestamps. The correctness of the protocol does not depend on the synchronization precision, though clock skew between servers can impact performance.

Similar to Detock, CAUSALDOT relies on deterministic transaction execution to substantially reduce cross-region coordination. Clients send transactions to their closest region. The first server that receives a transaction becomes its coordinator, which first resolves non-deterministic commands in the transaction (e.g. random() and time()), then attempts to extract its read/write set. When this is not possible via static analysis, the OLLP protocol is used, which obtains an initial estimate of the transaction's read/write set via a reconnaissance query [8]. Each region maintains a distributed index called a Home Directory that contains the cached value of the current home for each known data item. The Forwarder of the coordinator uses this index to augment the read/write set with the home information of every data item. It then annotates the transaction with this augmented read/write set and forwards the transaction to its home region(s).

Once these transactions reach their home regions, they are put into batches and inserted into a Paxos-maintained local log by the Sequencers. This log is synchronously replicated within a region to tolerate failure of individual servers, and optionally

to nearby regions to increase availability during (rare) failures of an entire region.

A region can deterministically replay a local log from any other region R to obtain the state of R's local data. Therefore, persisting the local logs is sufficient for durability, and replication is performed by shipping these local logs. While it is not required for a region to hold any remote data, having a possibly stale copy of the remote data allows local snapshot reads of data from other regions and makes executing multi-home transactions (including the home-movement transactions in Section 4) faster. To this end, regions in CAUSALDOT and Detock asynchronously exchange their local logs to each other, so each region eventually receives and replays the complete local log from every other region, as can be seen in the Log Managers of both regions.

If all data accessed by a transaction belong to a single region, it is called a single-home transaction; otherwise, it is multi-home. Multi-home transactions insert records into the local logs of each home region for the data accessed by that transaction. As mentioned above, Detock globally uses a global dependency graph for multi-home transactions to avoid inconsistently ordering them across regions (e.g. T1 before T2 at region 1, but T2 before T1 at region 2) which could result in serializability violations, OCC aborts, or deadlock. In contrast, CAUSALDOT annotates updates with the transaction commit time; a vector clock with an entry per region. Commit times produce a partial order that respects causal consistency.

CAUSALDOT retains the existing advantages of Detock. CAUSALDOT is able to guarantee that each transaction, regardless of its type, only needs a single-round trip from the initiating region to the participating regions: the initiating region sends the multi-home transaction to each region which houses data that it accesses, waits to receive the local log records back from those regions. By eliminating the global dependency graph, CAUSALDOT avoids the block caused by waiting to construct a complete dependency graph. Single-home transactions are non-blocking upon reaching their home region. Meanwhile, multi-home transactions can be executed once a home region has received messages from all other home regions, without the need to wait for the commit of its dependent transactions.

IV. PROTOCOLS OF CAUSALDOT

When a new transaction arrives at the system, its coordinator invokes the function StartTxn shown in Algorithm 1. The Home Directory is used to find the cached values of the home regions for all data items in the read/write set (Line 2-5).

Incorrect read/write sets (from the OLLP protocol) or home information (from stale values in the Home Directory) are deterministically detected later during execution and cause the transaction to abort and restart. However, these restarts are expected to be uncommon in practice: OLLP aborts only occur when the access set of data depends on the current state of the database [22], while home information aborts only occur for a short period of time after data is rehoused in a different region. The transaction is then forwarded to the participating regions (Line 6).

Algorithm 1 Starting a new transaction

```
1: function STARTTXN
2:     for (Key ∈ txn.ReadSets ∪ txn.WriteSets) do
3:         txn.homeinfo ← HomeDirectory(Key)
4:     end for
5:     txn.region = unique regions in txn.homeinfo
6:     send ⟨PrepareReq txn⟩ to all txn.region
7: end function
```

Algorithm 2 Scheduling transactions for execution

```
1: upon receive ⟨PrepareReq txn⟩ from client do
2:     txn.pt ← max{Clock, ct + 1}
3:     if (size of txn.region = 1) then
4:         ct = txn.pt
5:         UPDATE(txn, ct)
6:         Asynchronously send GlobeLog[txn] to all region
7:     else
8:         send ⟨CommitReq txn⟩ to all txn.region
9:     end if
10: upon receive ⟨CommitReq txn⟩ from all txn.region do
11:    pt ← ∅
12:    for (txn in ⟨CommitReq txn⟩) do
13:        pt ← max{pt, txn.pt}
14:    end for
15:    ct ← max{pt, ct}
16:    UPDATE(txn, pt)
17:    Asynchronously send GlobeLog[txn] to all region
```

A. Single-home Transactions

When a single-home transaction reaches a node at its presumed home region, the Sequencer of that node runs the code in Algorithm 2.

Unlike Detock, CAUSALDOT facilitates faster execution of single-home transactions. In Detock, the commit of single-region transactions must wait for all their dependent transactions to be completed. In contrast, in CAUSALDOT, single-region transactions can be committed non-blockingly. To ensure their deterministic ordering globally, CAUSALDOT annotates updates with the transaction commit time(Line 4); a vector clock with an entry per region. Commit times produce a partial order that respects causal consistency. The protocol uses these commit times to make transactions visible in accordance with causality. Single-home transactions are immediately visible to clients when they commit, as their causal dependencies are automatically satisfied.

The single-home transactions with timestamp will be asynchronously replicated from that region to every other region, such that every region eventually receives the complete set of ordered batches from every other region (Line 6). CAUSALDOT also support synchronous replication to near-by regions for improved robustness to region-failure as Detock. Unlike Detock, CAUSALDOT will never interleave transactions in different ways as all transactions will be ordered by their timestamps.

B. Multi-home Transactions

When a multi-home transaction reaches a participating region, it follows a different protocol than that of a single-home transaction. Upon receiving a multi-home transaction, the home region determines the transaction's prepare timestamp by taking the later timestamp between the current time and the region's commit timestamp(Line 2). This prepare timestamp is then sent to all transaction's home region(Line 8). Once the prepare timestamp messages from all home regions are received, by calculating the latest timestamps among all prepare timestamps, all home regions can asynchronously determine the same commit timestamp, after which the multi-home transaction can be executed and stored with that timestamp. Although the commit timestamp may be greater than the current timestamp, preemptive execution does not violate consistency. Subsequently, similar to single-home transactions, the timestamp will be asynchronously replicated from that region to every other region, ensuring that every region eventually receives the complete set of ordered batches from every other region. To maintain consistency, it is also necessary to adjust the region's commit timestamp to the transaction's commit timestamp.

In contrast, the dependency graph approach adopted by Detock for multi-home transactions can lead to significant latency. For example, if the home regions for multi-home transaction X are A and B, for Y are B and C, then it is necessary to wait until region B receives the dependency graph from region A, and region C receives the dependency graph from region D before transaction Y can be executed in region D. When there are many conflicting transactions, the latency problem will be severe. What's more, it may also lead to a livelock situation where it is challenging to obtain the complete dependency graph for an long time [11]. In comparison, multi-home transactions in CAUSALDOT can be executed immediately after one home region receiving messages from all other home regions.

C. Correctness

We now provide an informal proof sketch that CAUSALDOT provides causal consistency by showing that i) reads observe a causally consistent snapshot and ii) writes are atomic.

Proposition 1. *If an update u_2 causally depends on an update u_1, $u_1 \rightsquigarrow u_2$, then $u_1.ut < u_2.ut$.*

Proof: Let c be the client that wrote u_2. There are three cases upon which u_2 can depend on u_1, described in Section II-A: *1*) c committed u_1 in a previous transaction; *2*) c has read u_1, written in a previous transaction and *3*) c has read u_3, and there exists a chain of direct dependencies that lead from u_1 to u_3, i.e. $u_1 \rightsquigarrow ... \rightsquigarrow u_3$ and $u_3 \rightsquigarrow u_2$.

Case 1. When a client commits a transaction, it piggybacks the last update transaction commit time hwt_c, if any, to its commit request for the transaction coordinator (Alg. 1 Line 27) which is, furthermore, piggybacked as ht in its prepare requests to the involved partitions (Alg. 2 Line 23). To reflect causality when proposing a commit timestamp, each partition proposes higher timestamp than both ht and the snapshot timestamp (Alg. 3 Lines 10–14). The coordinator of the transaction chooses the maximum value from all proposed times from the participating partitions (Alg. 2 Line 26) to serve as commit time, ct, for all the updated items in the transaction. The new version of the data item is written in the key-value store with ct as its update time, ut (Alg. 4 Lines 2 and 13). When c commits the transaction that updates u_2, it piggybacks the commit time of the transaction that updated u_1. Hence, from the discussion above it follows that $u_1.ct < u_2.ct$. Because the commit time of a transaction is the update time of all the data item versions updated in the transaction (Alg. 4 Lines 2 and 13), we have $u_1.ut < u_2.ut$.

Case 2: c could have read u_1 either from c's client cache or from the transaction causal snapshot sn_T.

If c has read u_1 from c's client cache, then c has written u_1 either in a previous transaction in the same thread of execution or in the current one. If c wrote u_1 in the same transaction where u_2 is also written, then it is not possible to have $u_1 \rightsquigarrow u_2$ because all the updates from that transaction will be given the same commit, i.e. update timestamp, indicating that $u_1.ut = u_2.ut$. Thus, u_1 must be written in a previous transaction and from *Case 1* it follows that $u_1.ut < u_2.ut$.

Next, we will consider the case when c read u_1 from the causal snapshot sn_T that contains u_1. When a transaction T is started, the snapshot sn_T is determined by Alg. 2 Line 2, $sn_T = max\{ust_c, ust_n^m\}$. From Alg. 3 Line 5 we have that $u_1.ut \leq ust = sn_T$. From Lemma **??** it follows that $u_1.ut \leq sn_t < u_2.ct = u_2.ut$. Therefore, $u_1.ut < u_2.ut$.

Case 3: If u_2 depends on u_1 because of a transitive dependency out of $c's$ thread-of-execution, it means that there exists a chain of direct dependencies that lead from u_1 to u_2, i.e., $u_1 \rightsquigarrow ... \rightsquigarrow u_3$ and $u_3 \rightsquigarrow u_2$. Each pair in the transitive-chain, belongs to either *Case 1* or *Case 2*. Hence, the proof of *Case 3* comes down to chained application of the correctness arguments from *Case 1* and *Case 2*, proving that each element has an update time lower than its successor's.

■

Proposition 2. *Writes are atomic.*

Proof: Although updates are made visible independently on each region involved in the commit phase, either all updates are made visible or none of them are, i.e. the atomicity is not violated. All updates from a transaction belong to the same snapshot because they all receive the same commit timestamps(Alg. 2 Line 12–14). The updates are being installed in the order of their commit timestamps(Alg. 2 Line 16). The visibility of the item versions is determined by the transaction snapshot, which is based on the value of universal stable time.

■

V. RELATED WORK

Causal consistency database systems. Numerous works have proposed timestamp-based causal consistency models. GentleRain [14] and Eiger [23], along with its variants such as

Eiger-PORT [24] and EIGER-NOC2 [25], only support single read or write operations instead of transactions. Cure [26] necessitates global replication, whereas PaRiS [27] supports partial replication but requires two rounds for write transactions. Additionally, these models are not deterministic database system. In contrast, CAUSALDOT supports transactions and partial replication while only necessitating a single round trip from the initiating region to the participating regions.

Deterministic database systems. Previous works on deterministic databases have offered strict consistency, rendering their performance incomparable to that of CAUSALDOT. Furthermore, some blockchain protocols share characteristics similar to deterministic database systems [28] [29]. However, blockchain protocols do not impose requirements on data freshness, often only satisfying serializability instead of causal consistency, which limits their application in edge databases.

VI. CONCLUSION

We introduce CAUSALDOT, the first deterministic database model guaranteeing causal consistency. Compared to Detock, CAUSALDOT achieve non-blocking execution of single-region transactions and avoids the delay in constructing a complete cross-regional dependency graph by choosing a weaker consistency model. Future work will include an evaluation comparing the performance of CAUSALDOT with Detock and other protocols, where the performance advantage of CAUSALDOT is anticipated. We believe that CAUSALDOT will become a competitive protocol for edge databases and provide new ideas for the design of future deterministic databases.

REFERENCES

[1] Azure iot edge. https://azure.microsoft.com/en-gb/services/.

[2] Google cloud iot core. https://cloud.google.com/iot-core/.

[3] Regions, availability zones, and local zones. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/.

[4] I. Eyal, K. Birman, and R. Van Renesse, "Cache serializability: Reducing inconsistency in edge transactions," in *2015 IEEE 35th International Conference on Distributed Computing Systems*. IEEE, 2015, pp. 686–695.

[5] D. J. Abadi and J. M. Faleiro, "An overview of deterministic database systems," *Communications of the ACM*, vol. 61, no. 9, pp. 78–88, 2018.

[6] L. Lamport, "The part-time parliament," in *Concurrency: the Works of Leslie Lamport*, 2019, pp. 277–317.

[7] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX annual technical conference (USENIX ATC 14)*, 2014, pp. 305–319.

[8] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi, "Calvin: fast distributed transactions for partitioned database systems," in *Proceedings of the 2012 ACM SIGMOD international conference on management of data*, 2012, pp. 1–12.

[9] K. Ren, D. Li, and D. J. Abadi, "Slog: Serializable, low-latency, geo-replicated transactions," *Proceedings of the VLDB Endowment*, vol. 12, no. 11, 2019.

[10] F. Nawab, D. Agrawal, and A. El Abbadi, "Dpaxos: Managing data closer to users for low-latency and mobile applications," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1221–1236.

[11] C. D. Nguyen, J. K. Miller, and D. J. Abadi, "Detock: High performance multi-region transactions at scale," *Proceedings of the ACM on Management of Data*, vol. 1, no. 2, pp. 1–27, 2023.

[12] Tpc-c. https://azure.microsoft.com/en-gb/services/.

[13] A.-M. K. Pathan, R. Buyya *et al.*, "A taxonomy and survey of content delivery networks," *Grid computing and distributed systems laboratory, University of Melbourne, Technical Report*, vol. 4, no. 2007, p. 70, 2007.

[14] J. Du, C. Iorgulescu, A. Roy, and W. Zwaenepoel, "Gentlerain: Cheap and scalable causal consistency with physical clocks," in *Proceedings of the ACM Symposium on Cloud Computing*, 2014, pp. 1–13.

[15] M. Ahamad, G. Neiger, J. E. Burns, P. Kohli, and P. W. Hutto, "Causal Memory: Definitions, Implementation, and Programming," *Distributed Computing*, vol. 9, no. 1, pp. 37–49, 1995.

[16] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.

[17] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, "Don't Settle for Eventual: Scalable Causal Consistency for Wide-area Storage with COPS," 2011.

[18] J. Du, C. Iorgulescu, A. Roy, and W. Zwaenepoel, "GentleRain: Cheap and Scalable Causal Consistency with Physical Clocks," in *Proc. of SoCC*, 2014.

[19] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, "Stronger Semantics for Low-latency Geo-replicated Storage," in *Proc. of NSDI*, 2013.

[20] H. Lu, C. Hodsdon, K. Ngo, S. Mu, and W. Lloyd, "The SNOW Theorem and Latency-optimal Read-only Transactions," in *OSDI*, 2016.

[21] NTP, "The Network Time Protocol," http://www.ntp.org, 2017.

[22] K. Ren, A. Thomson, and D. J. Abadi, "An evaluation of the advantages and disadvantages of deterministic database systems," *Proceedings of the VLDB Endowment*, vol. 7, no. 10, pp. 821–832, 2014.

[23] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, "Stronger semantics for {Low-Latency}{Geo-Replicated} storage," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 2013, pp. 313–328.

[24] H. Lu, S. Sen, and W. Lloyd, "{Performance-Optimal}{Read-Only} transactions," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 333–349.

[25] S. Liu, L. Multazzu, H. Wei, and D. A. Basin, "Noc-noc: Towards performance-optimal distributed transactions," *Proceedings of the ACM on Management of Data*, vol. 2, no. 1, pp. 1–25, 2024.

[26] D. D. Akkoorath, A. Z. Tomsic, M. Bravo, Z. Li, T. Crain, A. Bieniusa, N. Preguiça, and M. Shapiro, "Cure: Strong semantics meets high availability and low latency," in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2016, pp. 405–414.

[27] K. Spirovska, D. Didona, and W. Zwaenepoel, "Paris: Causally consistent transactions with non-blocking reads and partial replication," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 304–316.

[28] Z. Lai, C. Liu, and E. Lo, "When private blockchain meets deterministic database," *Proceedings of the ACM on Management of Data*, vol. 1, no. 1, pp. 1–28, 2023.

[29] Z. Peng, Y. Zhang, Q. Xu, H. Liu, Y. Gao, X. Li, and G. Yu, "Neuchain: a fast permissioned blockchain system with deterministic ordering," *Proceedings of the VLDB Endowment*, vol. 15, no. 11, pp. 2585–2598, 2022.