Deterministic Concurrency Control For Private Blockchains

Ziliang Lai (Supervisor: Eric Lo)
The Chinese University of Hong Kong
{zllai,ericlo}@cse.cuhk.edu.hk

1 PROBLEM AND MOTIVATION
Private blockchain has gained popularity in inter-organization business processing (e.g., supply chain management [15] and cross-border payment [9]). It allows mutually untrusted parties to share data and mutate data in an endorsed manner. A typical private blockchain is composed of a consensus layer pipelined with an execution layer, where the organizations agree on the blocks of transactions using the consensus layer and the transaction blocks are then passed to each organization’s executor for execution. To ensure the organizations share an identical view of data, one core requirement of the executor is determinism, i.e., given the same transaction blocks and the same initial state, the executors must produce an identical new state.

While the consensus layer was regarded to be the major bottleneck of the blockchains, the relaxed requirement in the inter-organization setting (e.g., static membership, stable network connection) enables recent optimizations [3, 10, 23] to achieve astonishingly good performance, which in turn shifts the bottleneck to the execution layer [1, 20]. However, even the latest research in private blockchain offers unsatisfactory throughput in the execution layer due to the requirement of determinism. That’s because determinism limits the parallelism of execution since parallel execution often results in non-deterministic execution order. Therefore, recent works cannot execute transactions with full concurrency [1, 14, 18–20]. In this paper, we present Harmony, a deterministic concurrency control algorithm that is highly concurrent and yields $2.0 \times$ to $3.5 \times$ better throughput over the state-of-the-art competitors. Harmony is built on the shoulder of the latest deterministic databases [7, 8, 12, 21] that optimize concurrency control while upholding determinism. Based on that, Harmony incorporates inter-block parallelism and judicious abort which is crucial for performance in blockchain setting.

2 BACKGROUND AND RELATED WORK

2.1 Private Blockchains
Besides the work on optimizing the consensus layer [3, 6, 10, 11, 23, 25], the importance of the concurrency control in the execution layer has also been noticed [1, 14, 18–20]. Table 1 shows a summary of some representatives.

Serial execution is a straightforward method for ensuring determinism. Given the same transaction block generated from the consensus layer, the executor in Quorum [2] and Tendermint [4] executes the transactions serially following the order of their transaction IDs (TIDs). Assuming the transactions contain no non-deterministic operations (e.g., rand() and time()), each executor is guaranteed to yield the same state. However, this method yields limited throughput because it fails to utilize the core-parallelism in modern CPUs.

2.2 Deterministic Databases

While private blockchains have started to pursue concurrent transaction execution recently, deterministic databases [7, 8, 12, 21] have already studied deterministic concurrency control for almost a decade. Astonishingly, the connection between private blockchain and deterministic database has never been explicit, although the connection between the former and distributed database has been

<table>
<thead>
<tr>
<th>Private Blockchains</th>
<th>Execution Layer</th>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quorum [2]</td>
<td>Serial execution</td>
<td>No concurrency</td>
</tr>
<tr>
<td>Tendermint [4]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fabric [1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fabric++ [20]</td>
<td>(1) Parallel simulation</td>
<td>Limited concurrency</td>
</tr>
<tr>
<td>Fabric* [18]</td>
<td>(2) Serial validation</td>
<td></td>
</tr>
<tr>
<td>RBC [14]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: The problem of the execution layer in existing private blockchains

![Figure 1: The problems of applying the latest deterministic database to blockchain, and our proposed methods](image-url)

Some recent private blockchains employ optimistic methods to enable partial parallelism [1, 14, 18, 20]. Specifically, the transactions are executed in a Simulation phase and a Validation phase. The Simulation phase is able to execute transactions optimistically in parallel, while the Validation phase has to validate serializability (i.e., the concurrent execution is equivalent to a serial execution) in a serial manner to uphold determinism. Therefore, the degree of concurrency is limited.

![Figure 1: The problems of applying the latest deterministic database to blockchain, and our proposed methods](image-url)
studied elsewhere [17]. Private blockchain and deterministic database actually share a lot in common. First, both of them are distributed in nature. Second, they both process transactions block-by-block. Third, both blockchains and deterministic databases require determinism.

One reason that impedes deterministic databases to be adopted to private blockchains is the strong assumptions required by the classic deterministic databases. They assume every transaction’s read-write sets can be inferred before execution [7, 8, 21], but that is not generally applicable (e.g., when the transaction contains branches based on the query results). Nonetheless, the latest deterministic database Aria [12] eliminates this requirement by executing transactions in a Simulation phase and a Validation phase similar to the recent private blockchains. However, Aria is more advanced than private blockchains by offering concurrency in both phases.

Given a block of transactions, Aria generates deterministic read-write sets of the transactions in the Simulation phase. Taking the read-write sets as input, its Validation phase validates serializability by examine dangerous dependency patterns and commits transactions deterministically in parallel. Concretely, in the Simulation phase, Aria executes the transactions against a block snapshot, which is the state after processing the previous block. In this phase, no modification is applied, and each partition only buffers its write-set in its private workspace. With the same block snapshot and the same transaction blocks, executors in Aria generates identical read-write sets. In the Validation phase, the dependencies among transactions can be identified. For example, if two transactions \( T_1 \) and \( T_2 \) have overlapping write-sets, there is a write(w)-write(w)-dependency among them. Aria deterministically abort some transactions to uphold serializability by examine the dangerous dependency patterns. For example, the ww-dependency is one of the dangerous dependency patterns in Aria. On seeing \( T_1 \) ww-depending on \( T_2 \), Aria aborts the one with the larger TID (i.e., \( T_2 \)). By validating based on the deterministic read-write sets and the TIDs, a deterministic set of the transactions is aborted in this phase. Moreover, since the transactions that passes the validation have non-overlapping write-sets, they can apply their updates in parallel without breaking determinism.

However, even the latest deterministic database is not a perfect fit for private blockchain because of the difference in design choices. Specifically, Aria is designed for main-memory databases, while private blockchains are commonly disk oriented. That’s because disk-based storage is more cost effective and upgrading to main-memory databases is cheap— which is not the case in private blockchains because retrying a transaction means going through consensus and disk I/Os again.

- **Spurious Aborts.** Aborting a transaction on seeing a ww-dependency is overly conservative because two transactions \( T_1 \) and \( T_2 \) that update the same item \( x \) can still form a serializable schedule. While spurious aborts are acceptable in Aria because retrying an aborted transaction in main-memory databases is cheap— which is not the case in private blockchains because retrying a transaction means going through consensus and disk I/Os again.

\[ T_1 \xrightarrow{rw} T_2 \]

(a) A match with only two transactions \( T_1 \xrightarrow{rw} T_2 \xrightarrow{rw} T_3 \)

(b) A match with at least three transactions

Figure 2: Examples of backward dangerous structure

- **Stragglers.** Aria has strict barriers between blocks, i.e., only after block \( i \) has finished execution, block \( (i + 1) \) can start, which causes the straggler problem. Due to disk stalls, the variance of the transaction runtime in private blockchains is higher than main-memory databases, and thus the straggler problem is exaggerated.

3 **UNIQUENESS AND APPROACH**

3.1 Fusing Deterministic Database With Private Blockchain

**Insight.** Given the close connections, private blockchains can actually absorb many techniques from deterministic databases. In particular, the advanced deterministic concurrency control can be adopted to improve the concurrency in private blockchains.

**Approach.** We port the deterministic concurrency control algorithm in Aria to BCR [14], a private blockchain that builds on top of PostgreSQL [13], which allows blockchains to support full-fledged SQL functionalities like a relational database. To mitigate Aria’s downsides when applied to blockchain (i.e., spurious aborts and stragglers) we propose a new deterministic concurrency control Harmony, which incorporates blockchain-specific optimizations (see below).

3.2 Judicious Validation

**Insight.** The idea of dangerous dependency pattern enables transactions to validate independently in parallel because they only examine dependencies that is “local” to themselves. However, a pattern based on a single dependency (e.g., ww-dependency in Aria) could cause many spurious aborts. Harmony design a new dangerous patterns based on a pair “local” dependencies, such that fewer transactions match the pattern and abort, while the level of concurrency is almost unaffected.

**Approach.** Harmony abort a transaction if it matches a backward dangerous structure. Specifically, \( T_j \) is aborted if there exists \( T_i \xrightarrow{rw} T_j \xleftarrow{rw} T_k, \) \( i < j \) and \( i \leq k \), where we use \( \xleftrightarrow{rw} \) to represent read(r)-write(w)-dependencies.

Figure 2 shows two examples that match the backward dangerous structure. Notice that a backward dangerous structure can be as small as having two transactions only since we allow \( i = k \) (Figure 2a). A dangerous structure can also involve an arbitrary number of transactions (e.g., Figure 2b, where \( T_1 \xrightarrow{rw} T_3 \xrightarrow{rw} T_4 \)). Overall, by eliminating all backward dangerous structures, Harmony eliminates cycles in the rw-dependency graph. Nonetheless, all the cycles in the complete dependency graph must be eliminated to achieve serializability. Harmony offloads the ww-dependencies to update reordering such that the whole dependency graph is acyclic.
3.3 Update Reordering

**Insight.** Given an acyclic rw-dependency graph, reordering the rw-dependencies based on the topological order of the rw-dependency graph allows the complete dependency graph to be acyclic. As the example shown in Figure 1, by reordering the rw-dependencies following the direction of the rw-dependencies, the backward edges are eliminated and thus cycles are avoided in the dependency graph.

**Approach.** Although useful, the topological sort is expensive and hard to parallelize. We exploit the property of the acyclic rw-dependency graph (i.e., all backward dangerous structures are removed) to convert the topological sort into quicksorts.

Figure 3 shows an example graph without backward dangerous structures. A topological sort on it results in \( \{T_4, T_1, T_2, T_3\} \). Harmony computes a \( \text{min}_{\text{out}} \) for each transaction \( T \) representing the minimum TID in \( T \)'s outgoing edges. Suppose only \( T_2 \) and \( T_4 \) update \( x \), Harmony only needs to quick-sort the \( \text{min}_{\text{out}} \)s of \( T_2 \) and \( T_4 \) without traversing the whole graph like topological sort. Nonetheless, the resulting order (i.e., \( \{T_4, T_2\} \)) is consistent with the topological ordering. Besides, if \( T_1 \) and \( T_3 \) update \( y \), the two updater lists (i.e., \( \{T_2, T_4\} \) on \( x \) and \( \{T_1, T_2, T_3\} \) on \( y \)) can be sorted in parallel.

4 RESULTS AND CONTRIBUTIONS

4.1 Performance Comparisons

We perform experiments on machines with 2.1GHz Intel Xeon E5-2620v4 with 64GB DRAM and 800GB SSD, running 64-bit CentOS 7.6 with Linux Kernel 3.10.0 and GCC 4.8.5. Nodes are connected using 1Gbps Ethernet. Harmony is compared with state-of-the-art blockchains: Fabric [1], FastFabric [18], and RBC [14]. Besides, to demonstrate the effectiveness of our blockchain-specific optimizations, we also compared with AriaBC, i.e., the blockchain that we built by directly porting the deterministic concurrency control in Aria to RBC [14]. There are four organizations in our experiment and we follow the recent works [1, 14, 18, 20] to use Kafka as the consensus layer.

We use both YCSB [5] and Smallbank [16] as the benchmarks. For YCSB, we set the number of keys to 10k and follow [12, 22, 24] to wrap 10 operations into one transaction. Each operation has equal probabilities of being a read (SELECT) or a write (UPDATE). For Smallbank, we also set the number of accounts to 10k and use the standard mix. Both benchmarks have a skewness of 0.6 by default to create medium contention.

We measure the peak throughput and the end-to-end latency of the committed transactions for all systems. Figures 6 and 7 show the results. HarmonyBC attains 3.5K and 2.0K throughput over the best of the existing private blockchains (i.e., RBC in this experiment) in Smallbank and YCSB, respectively. It also achieves the latency comparably good as the best of the state-of-the-art.
Simply implementing Aria as a private blockchain (i.e., AriaBC) readily yields better throughput than existing private blockchains, which shows the benefits of using an advanced deterministic concurrency control from a deterministic database to improve private blockchain. Harmony attains a larger margin (e.g., 1.5× throughput over AriaBC in YCSB), demonstrating the effectiveness of Harmony’s blockchain-specific optimizations.

To further compare the performance under different contentions, we vary the skewness of the accessed data to control the degree of contention. As shown in Figures 8 and 9, all systems incur more aborts with a larger skewness and thus their performances drop (less severe in Smallbank since it generally has lower contention compared to YCSB). But, Harmony outperforms its competitors under all skewnesses and it consistently has lower abort rates.

4.2 Contributions

This paper makes the following major contributions:

- We take the first step to explicitly connect private blockchain with deterministic database. By exploiting the connections, we are able to fast-forward the development of private blockchain using lessons learned from deterministic databases.
- Based on the state-of-the-art deterministic concurrency control in the latest deterministic database, we proposed effective techniques to further boosts its performance in blockchain setting.
- Harmony is extensively evaluated using benchmarks commonly used in blockchains including YCSB [5] and SmallBank [16]. Empirical results show that Harmony achieves 2.0× to 3.5× throughput better than RBC [14] and FastFabric# [18], and 2.3× throughput better than AriaBC (the blockchain implemented using the same framework as Harmony but using Aria) under high contention.

REFERENCES


