Slide 1 Start

Good morning everyone. I'm Yangna from nanjing university. Today, I'm excited to present our ICDE 2025 paper titled "Boosting End-to-End Database Isolation Checking via Mini-Transactions".This is a joint effort by researchers from Nanjing University, ETH, and Tencent.

Slide 2: Transaction and Isolation Level

Let's start with some background.

We define a transaction as a pair consisting of a set of operations and a strict total order over them, which we call the program order.Transactions are issued by clients and grouped into sessions—each session being a sequence of transactions.

In transactional databases, isolation levels define how concurrently executed transactions interact with each other.They control the visibility of intermediate results and help prevent anomalies such as dirty reads, lost updates, or write skew.

Slide 3: Serializability (SER)

Our approach is mainly for strong isolation levels, including strict serializability,serializability, and snapshot isolation.We will take serializability as an example.

Serializable is the strongest widely implemented isolation level.

It guarantees that the outcome of concurrent transactions is equivalent to some serial execution.

As shown in the figure, on the surface, concurrent transactions are interleaving with the database, but in fact, concurrent transactions are executed serially.

Slide 4: Database Systems and Serializability

However, recent studies have uncovered numerous isolation bugs in many production databases, raising concerns about whether these databases actually uphold their promised isolation guarantees in practice. Isolation bugs are very difficult to detect, as they often produce incorrect results without any explicit errors.

The above pictures are all reports of databases that violate the Serializability isolation level found by Jepsen.

Slide 5: Black-box Checking(End to End)

Black-box database isolation checking involves two stages: history generation (Step 1 , 2 ,and 3) and history verification (Step 4).

First, clients send transactional requests to the database(Step 1), treating it as a black-box system.

Each client records the requests sent to the database along with the corresponding results received (Step 2).

The logs from all clients are combined into a single history, which is provided to the isolation checker (Step 3).

Finally, the checker performs history verification, deciding whether the history satisfies the specified isolation level. If isolation violations are detected, some checkers offer counterexamples (Step 4).

Slide 6: General Transaction (GT)

General transaction workloads are widely used in randomized isolation testing, but they face serious efficiency challenges.

First, executing many concurrent GTs creates high overhead—especially under strong isolation levels—due to frequent transaction aborts or costly locking.This will result in a large amount of checking overhead during history generation.

Second, GT histories produce large and dense dependency graphs, which are expensive to verify.This will result in a large amount of checking overhead during history verification.

Slide 7 :Reseach Question:

So how to boost end-to-end database isolation checking ?

We address these inefficiencies through the novel design of Mini-Transactions. MTs are compact, short transactions that execute much faster than general workloads, reducing overhead during history generation by minimizing database blocking and transaction retries. By leveraging MTs'read-modify-write pattern, we develop highly efficient algorithms to verify strong isolation levels in linear or quadratic time.

Slide 8:Read-Modify-Write pattern

However, even with these improvements in execution time,efficient verification against strong isolation levels remains challenging due to the potential for large transaction numbers,which generate dense dependency graphs.

1、To address this issue, we design MTs to adhere to the read-modify-write(RMW) pattern, where each write operation is preceded by a read operation on the same object.

2、since each write operation is preceded by a read operation on the same object in mini-transactions, the write-write conflict is almost determined.

3、Despite the NP-hardness of verifying strong isolation levels for general histories , we enable highly efficient algorithms in linear or quadratic time to verify strong isolation levels over MT histories，thanks to the read-modify-write pattern in MTs and the unique value conditions in MT histories.

Slide 9 : Motivation

Our insight behind the MT-based description of anomalies is that, while isolation violations have been detected in GT workloads of substantial size, these violations are typically due to a small subset of operations within a limited number of transactions. For example, Figure illustrates an SI violation identified in MariaDB.The corresponding GT workload comprises 10 sessions, each containing 100 transactions, with each transaction performing 10 operations. However, as highlighted in red, the core of this bug involves only three transactions, each with two operations,which can be represented by our MTs.

Slide 10: Contributions

Our contributions are three-fold:

First, at the conceptual level, we propose a unified MT-based approach that addresses inefficiencies in both history generation and verification in black-box isolation checking.

Second, at the technical level, we design Mini-Transactions that are compact yet semantically rich —capable of capturing all 14 well-known isolation anomalies.We develop efficient verification algorithms using the read-modify-write pattern and unique values, and prove that verifying strong isolation without unique values is NP-hard.

Third, at the practical level, we implement our techniques in a tool called MTC, which includes both an MT workload generator and verifier.Experiments show that MTC significantly outperforms state-of-the-art tools in end-to-end checking, and effectively detects real-world bugs across isolation levels.

Slide 11: Mini-Transactions

A mini-transaction is a transaction meeting the following two rules:

• It contains one or two read operations and at most two write operations.

• Each write operation, if present, is preceded by a read operation on the same object.

A mini-transaction history is a history comprising solely mini-transactions such that each write operation on the same object assigns a unique value.

The first rule of mini-transactions helps expedite database execution during history generation, while the second rule, along with the unique values written, guarantees the existence of efficient verification algorithms for mini transaction histories.

Slide 12: MTs are highly expressive to characterize all common isolation anomalies.

Despite their simplicity, MTs are semantically rich enough to capture common isolation anomalies that can occur in GT histories. These encompass all 14 well-known isolation anomalies.For example, common anomalies such as LostUpdate and WriteSkew, and more complex anomalies such as CausalityViolation.

Slide 13: Dependency Graph based Characterization of SER

As before, we will take serializability as an example.A history satisfies SER, if and only if it satisfies INT and one of its dependency graph which includes SO edge,WR edge,WW edge and RW edge is cycle-free.

Now, let's look an example.

Slide 14: Dependency Graph based Characterization of SER

As shown in the figure,Transaction T0 writes the acct object, and then transactions TA and TB respectively read the acct object written by T0 . We say that TA and TB read-depends on T0 . We will add WR dependency edges from T0 to TA and T0 to TB in the Dependency Graph.

Slide 15: Dependency Graph based Characterization of SER

T0 write the acct object, and then TA and TB respectively wirte the acct object written by T0 . We say that TA and TB wirte-depends on T0 . We will add WW dependency edges from T0 to TA and T0 to TB in the Dependency Graph.

Slide 16: Dependency Graph based Characterization of SER

TA reads the acct object from T0 and then the object TA reads is overwritten by TB . We say that TB anti-depends on TA . We will add RW dependency edges from TA to TB in the Dependency Graph.

Slide 17: Dependency Graph based Characterization of SER

Similarly,TB reads the acct object from T0 and then the object TB reads is overwritten by TA . We say that TA anti-depends on TB . We will add RW dependency edges from TB to TA in the Dependency Graph.

Slide 18: Dependency Graph based Characterization of SER

At this point, undesired cycle for SER: TB anti-depends on TA and TA anti-depends on TB will occur.

Slide 19: Dependency Graph based Characterization of SER

We will continue to perfect the Dependency Graph. We assume that TA writes the acct object first and then TB overwrites the acct object. We say that TB write-depends on TA . We will add WW dependency edges from TA to TB in the Dependency Graph.

At this point, undesired cycle for SER: TB write-depends on TA and TA anti-depends on TB will occur.

Slide 20: Dependency Graph based Characterization of SER

Next,We assume that TB writes the acct object first and then TA overwrites the acct object. We say that TA write-depends on TB . We will add WW dependency edges from TB to TA in the Dependency Graph.

At this point, undesired cycle for SER: TA write-depends on TB and TB anti-depends on TA will occur.

Slide 21: Dependency Graph based Characterization of SER

We have considered both bases TB write-depends on TA and TA write-depends on TB and each case leads to an undesired cycle for SER.We draw a conclusion that this history doesn't satisfy SER.

Slide 22：Mini-Transaction History Verification Algorithm

We will take serializability as an example to analyze the verification algorithm.

The algorithm construct the dependency graph G of the input history H based on the dependency relations SO, WR, WW, and RW by calling BuildDependency.

Due to unique values written, the WR dependency edges are entirely determined.

The WW dependency edges are inferred from the WR dependency.

Finally, the RW dependency edges are added based on the WR and WW dependencies.

Subsequently, we check whether the dependency graph G is cycle-free for SER.

Slide 23：Algorithm Correctness

Our verification algorithms achieve linear or quadratic time for problems that are generally NP-hard. To justify this, we introduce formal correctness proofs. These proofs ensure that our optimizations do not compromise soundness or completeness. The correctness proofs of these three verification algorithms are surprisingly intricate and are essential to support our claim that "verifying strong isolation levels without unique values is NP-hard" while our methods remain tractable under MT constraints.

In addition，as shown in the figure, although the dependency graph contains a cycle that violates SER, this cycle is allowed under SI. To correctly detect SI violations, CHECKSI first detects the divergence pattern, which appears in this case. This shows that simply checking cycle-free is not enough for SI.

Slide 24: Optimized Verification Algorithm

We optimize our dependency graph construction by skipping the transitive closure of WW edges.

This reduces unnecessary edges and improves performance without trading off correctness.

This simple yet effective improvement significantly reduces graph size and runtime overhead.

Slide 25：Complexity Issues

Suppose that the input history comprises n transactions,and the dependency graph G returned by the BuildDependency procedure contains m edges. Given that a mini-transaction may have at most n incoming RT edges,one incoming SO edges, two incoming WR edges, two incoming WW edges, and two incoming RW edges, we can deduce dependency graph has $O(n)$ edges. Therefore, the time complexity of both the CHECKSER and CHECKSI procedures is Θ(n)(Theta of n).

Moreover, since establishing the RT edges requires Θ(n2)(Theta of n squared) time, the time complexity of CHECKSSER is Θ(n2)(Theta of n squared).

Slide 26：Experimental Evaluation

This section presents a comprehensive evaluation of MTC and state-of-the-art black-box isolation checkers that test databases under highly concurrent GT workloads. We aim to address the following four questions:

Slide 27：Q1: History Verification Performance vs. Cobra

MTC-SER consistently outperforms Cobra's GPU accelerated version across various concurrency levels. For MT histories, Cobra exhibits similar overhead without GPU acceleration. Under highly skewed object access patterns,MTC-SER achieves about 5x(five times) better performance than Cobra (Figure a).

Additionally, MTC-SER maintains stable performance with respect to skewness, while Cobra's verification time increases exponentially with fewer objects; see Figure b.

Both checkers maintain stable performance with respect to the number of sessions, yet MTC-SER consistently achieves around 2x(two times) better performance than Cobra (Figure c).

Furthermore, as the number of transactions increases, Cobra's verification time escalates significantly faster (Figure d).

Slide 28：Q1: History Verification Performance vs. PolySI

Compared to PolySI in verifying SI, MTC-SI demonstrates significantly greater performance gains across various work loads. MTC-SI can achieves about 1600x(one thousand six hundred times) faster verification (Figure a).

With an increasing number of transactions,MTC-SI reduces verification time by up to 93x(ninety-three times) (Figure d).

Slide 29：Q1: History Verification Performance vs. Porcupine

As shown in Figure, MTC-SSER outperforms porcupine across various concurrency levels. Under extreme concurrency where all clients execute transactions at the same time, MTC SSER demonstrates a substantial 28x(twenty-eight times faster) improvement in verification.

Furthermore, MTC-SSER maintains stable performance as the number of concurrent sessions increases (Figure a).

Slide 30：Q2: End-to-End Checking Performance vs. Cobra

To assess the performance improvement that MTs bring to the end-to-end isolation checking process, we compare base line checkers using their standard GT workload generators.

As shown in Figures a-c, MTC-SER with MT workloads substantially and consistently outperforms Cobra with GT workloads in both history generation and verification. Achieving up to two and three orders of greatness improvement in history verification compared to Cobra with and without GPU, respectively. Additionally, as concurrency levels increase, MT workload generation becomes increasingly more efficient compared to Cobra.

Figures d-f show that MTC-SER consistently consumes significantly less memory than Cobra, achieving up to thirty times improvement compared to Cobra with GPU, and up to fourteen times without GPU.

Slide 31：Q2: End-to-End Checking Performance vs. PolySI

As shown in Figures,we observe similar trends in both time and memory in the end-to-end isolation checking comparison between MTC-SI and PolySI.

Slide 32：Q3: Effectiveness of Workload Generation

Figure presents the abort rates for executing MT and GT workloads in PostgreSQL under SER and SI. For GT workloads, we use a moderate transaction size of 20 operations.

GT workloads result in substantially more aborted transactions, leading to less effective histories. As shown in Figure a, even with only a few client sessions, nearly half of the transactions are aborted. The abort rate increases as more sessions are involved, indicating greater stress on PostgreSQL. Moreover, the GT workload generator is sensitive to access skewness. In particular, when roughly 20 transactions access the same object, over 60% of transactions are aborted, as shown in Figure b. In contrast, our MT workload generator demonstrates robustness against variations in both cases.

Slide 33：Q4: Detecting Isolation Bugs - Rediscovering Bugs

MTC can successfully (re)discover real-world bugs across different isolation levels with MTs. For example, Figure a shows the LOSTUPDATE anomaly found in MariaDB which violates the claimed SI. Figure b describes the WRITESKEW anomaly found in PostgreSQL, which violates the claimed SER. Notably,the counterexamples returned by MTC are relatively concise and easy to interpret with MTs.

Slide 34：Q4: Detecting Isolation Bugs - Effectiveness

As shown in Figure a, within 30 minutes, MTC detects bugs in 32 successful trials on PostgreSQL, while Elle achieves the highest bug detection rate with 35 successful trials at a maximum transaction length of 8 operations under the list append workload.

On the other hand, MTC consistently detects bugs in more successful trials than Elle under the read-write register workload. On MongoDB (Figure b), MTC is also highly competitive in bug detection compared to Elle.

Slide 35：Q4: Detecting Isolation Bugs - Effectiveness

We also report the average time for history generation and verification for MTC and Elle. Figure shows that MTC consistently outperforms Elle in both tasks. For example, on PostgreSQL with a maximum transaction length of 8 operations under a read-write register workload, MTC achieves up to 3.5x(three point five times) and 15x(fifteen times) speedups in history generation and verification, respectively.

Slide 36：Summary

In summary,we propose a unified MT-based approach for efficient history generation and verification.

We design compact Mini-Transactions that capture all 14 isolation anomalies, and prove strong isolation checking without unique values is NP-hard.

We build the MTC tool, which outperforms existing checkers and detects real-world bugs across isolation levels.

Slide 37：Discussion & Future work

In our discussion,we observe that Elle's effectiveness in bug detection is highly sensitive to transaction sizes across databases, whereas a transaction size of 4 serves as a baseline.

A key challenge is to accurately collect transaction start and finish wall-clock timestamps, while handling potential clock skew.

As for future work, We plan to evaluate MTC's performance in verifying SSER by conducting experiments on FaunaDB and VoltDB.

Furthermore,We aim to extend MTC to support weaker isolation levels and even predicate-based verification, to further broaden its applicability.

Slide 38 End

Thank you for your attention. I welcome your questions.