TxnSails: Achieving Serializable Transaction Scheduling with Self-Adaptive Isolation Level Selection

Qiyu Zhuang[†], Wei Lu[†], Shuang Liu[†], Yuxing Chen[‡], Xinyue Shi[†]
Zhanhao Zhao[‡], Yipeng Sun[†], Anqun Pan[‡], Xiaoyong Du[†]

[†]Renmin University of China [‡]Tencent Inc.

04/09/2025

Tencent

Outline

Tencent

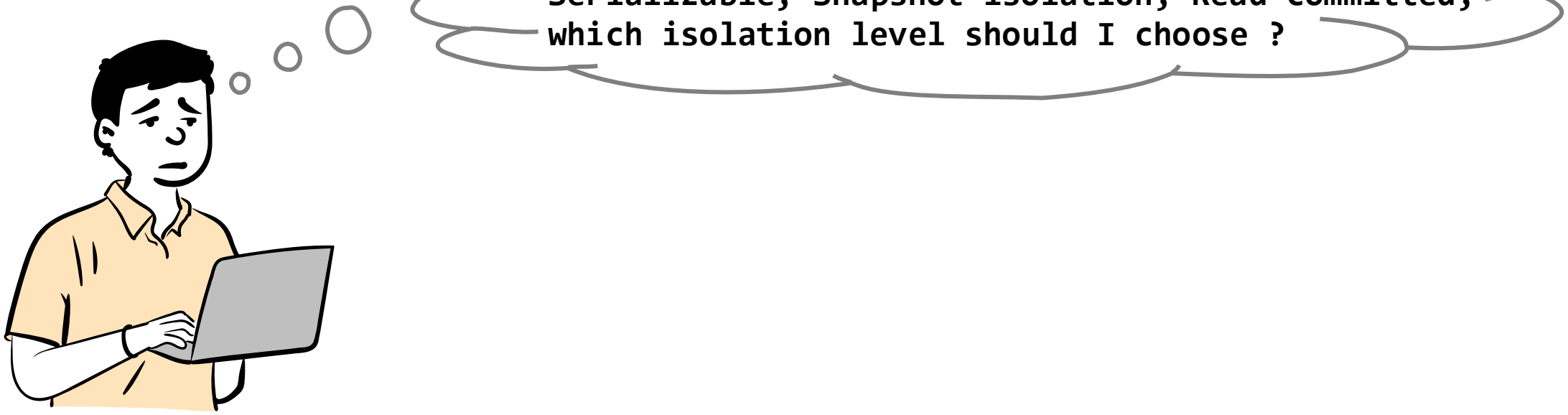
- Background
- System Overview
- Technique Details
- Evaluations
- Conclusion

Outline

- **Background**
- System Overview
- Technique Details
- Evaluations
- Conclusion

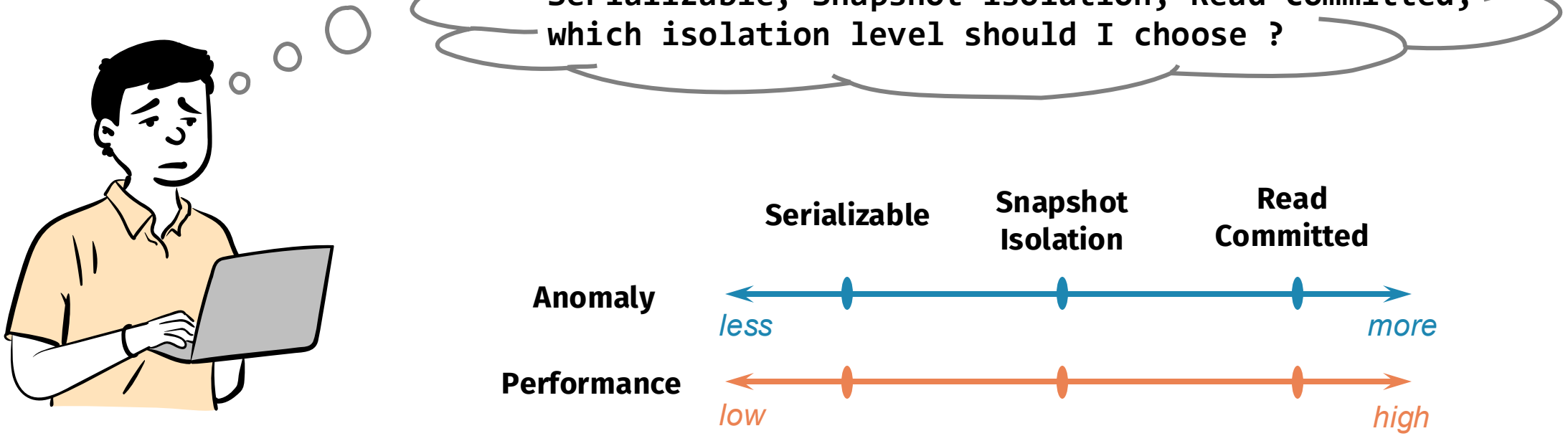
Background

- Applications rely databases to store and manage their data.



Background

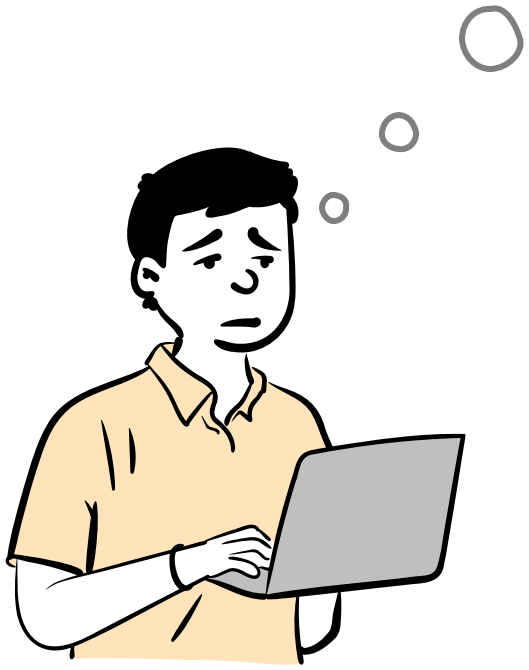
- Applications rely databases to store and manage their data.



Background


I need both high performance and no data anomaly,
what can I do without kernel modification ?

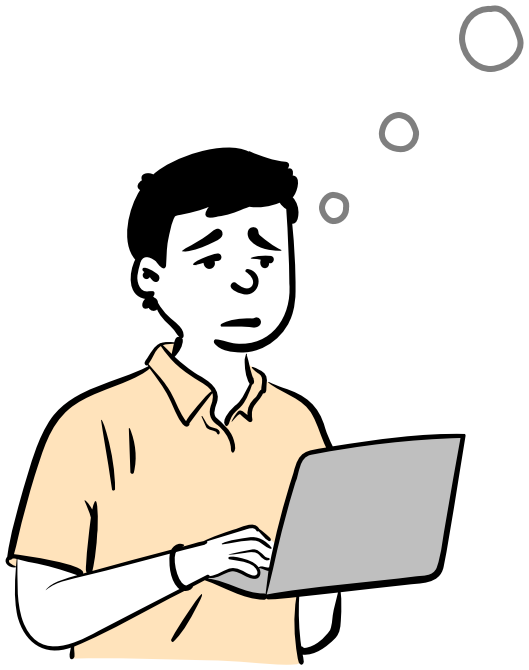
Background



I need both high performance and no data anomaly,
what can I do without kernel modification ?

Configure the database to a low isolation level.

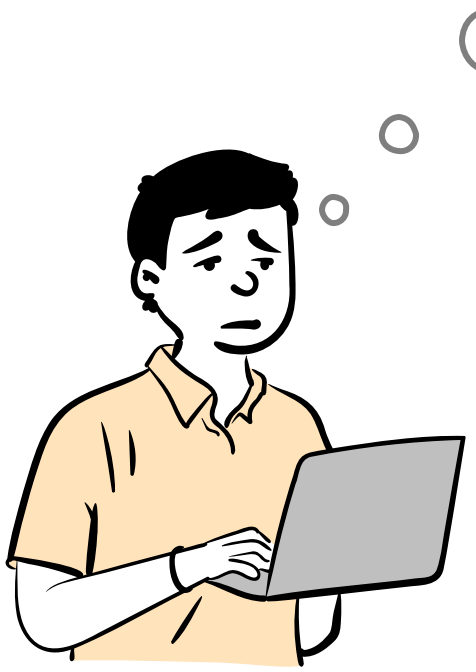
Background



I need both high performance and no data anomaly,
what can I do without kernel modification ?

How to avoid data anomalies when the database is configured to low isolation levels ?

Background



I need both high performance and no data anomaly,
what can I do without kernel modification ?

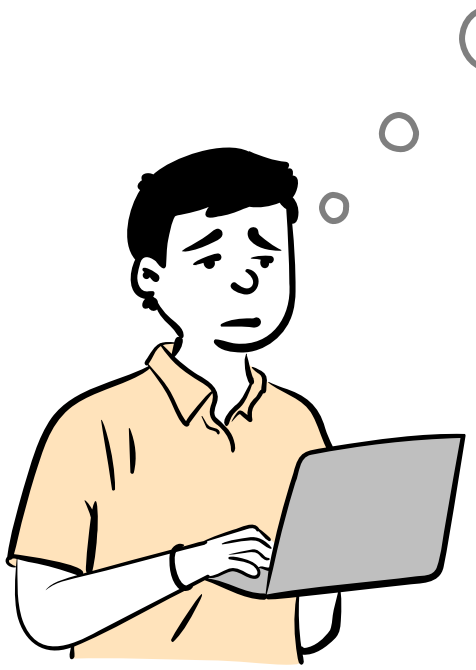
How to avoid data anomalies when the database is configured to low isolation levels ?

I. Do concurrency control inside the application [1].

High Performance but unsafe, easy to write critical bugs[1]

[1] Ad Hoc Transactions in Web Applications: The Good, the Bad, and the Ugly

Background



I need both high performance and no data anomaly,
what can I do without kernel modification ?

How to avoid data anomalies when the database is configured to low isolation levels ?

I. Do concurrency control inside the application [1].

II. Promote some specific reads to writes[2,3].

High Performance but overclaimed, prone to false positives

[1] Ad Hoc Transactions in Web Applications: The Good, the Bad, and the Ugly
[2] Serializable Use of Read Committed Isolation Level
[3] Robustness against Read Committed for Transaction Templates

Challenges

Tencent

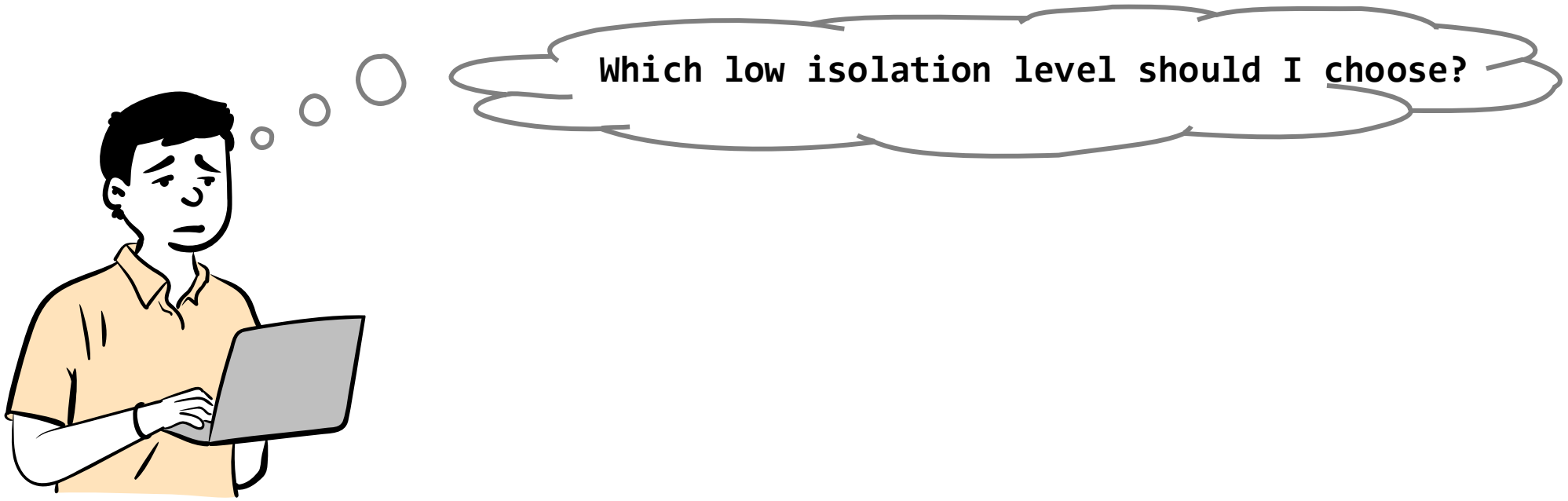
Challenge1:

How to elevate low isolation levels to SER without additional writes ?

Challenges

Challenge1:

How to elevate low isolation levels to SER without additional writes ?



Challenges

Tencent

Challenge1:

How to elevate low isolation levels to SER without additional writes ?

Challenge2:

How to determine the optimal isolation level for specific workload ?

Challenges

Tencent

Challenge1:

How to elevate low isolation levels to SER without additional writes ?

Challenge2:

How to determine the optimal isolation level for specific workload ?

Challenge3:

How to guarantee SER during the transition between isolation levels ?

Background

Data Anomaly Structure

Under SI, a **dangerous structure is** defined as the RW-dependency chain, $T_i \xrightarrow{rw} T_j \xrightarrow{rw} T_k$. In a data anomaly, at least one dangerous structure exists, and ***T_k commits before T_j commits***.

Under RC, a single RW-dependency, $T_i \xrightarrow{rw} T_j$, is referred to as a **dangerous structure**. In a data anomaly, at least one dangerous structure exists, and ***T_j commits before T_i commits***.

Vulnerable dependency

1. The RW dependency $T_j \xrightarrow{rw} T_k$ in chain $T_i \xrightarrow{rw} T_j \xrightarrow{rw} T_k$ under SI.
2. The single RW dependency $T_i \xrightarrow{rw} T_j$ under RC.

Necessary condition of anomaly

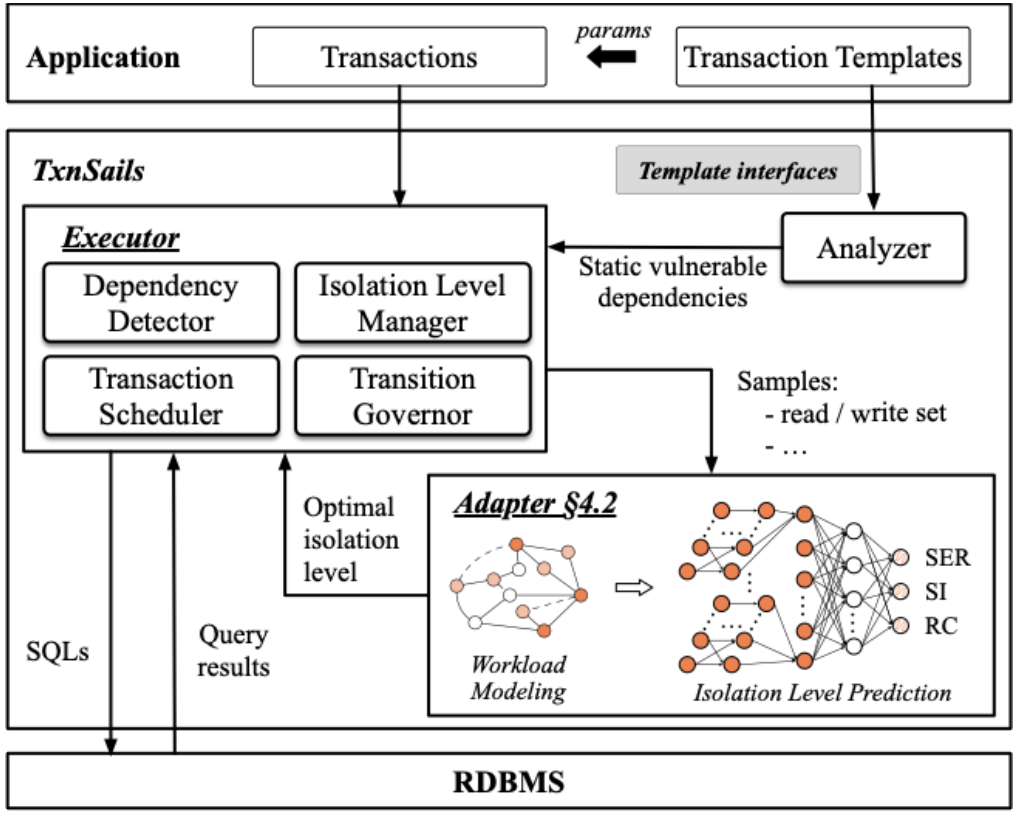
1. A vulnerable dependency exists.
2. The **dependency order** between the two transactions in the vulnerable dependency is inconsistent with their **commit order**.

Outline

- Background
- **System Overview**
- Technique Details
- Evaluations
- Conclusion

System Overview

TxnSails works in the middle tier between the application tier and database tier, it comprises three main components: ***Analyzer***, ***Executor***, and ***Adapter***.

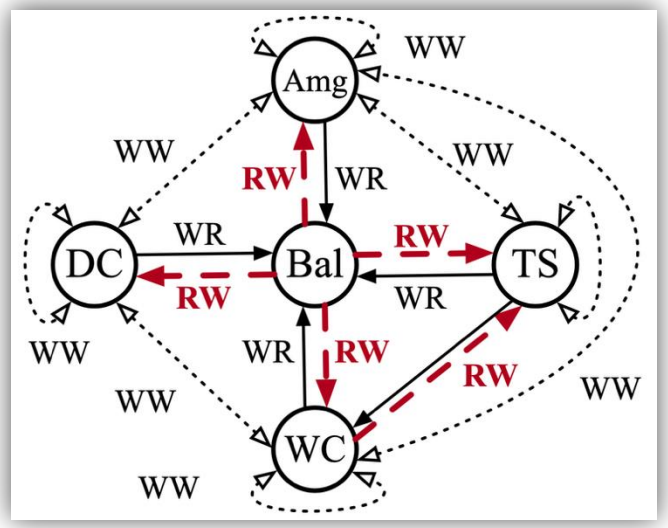


System Overview

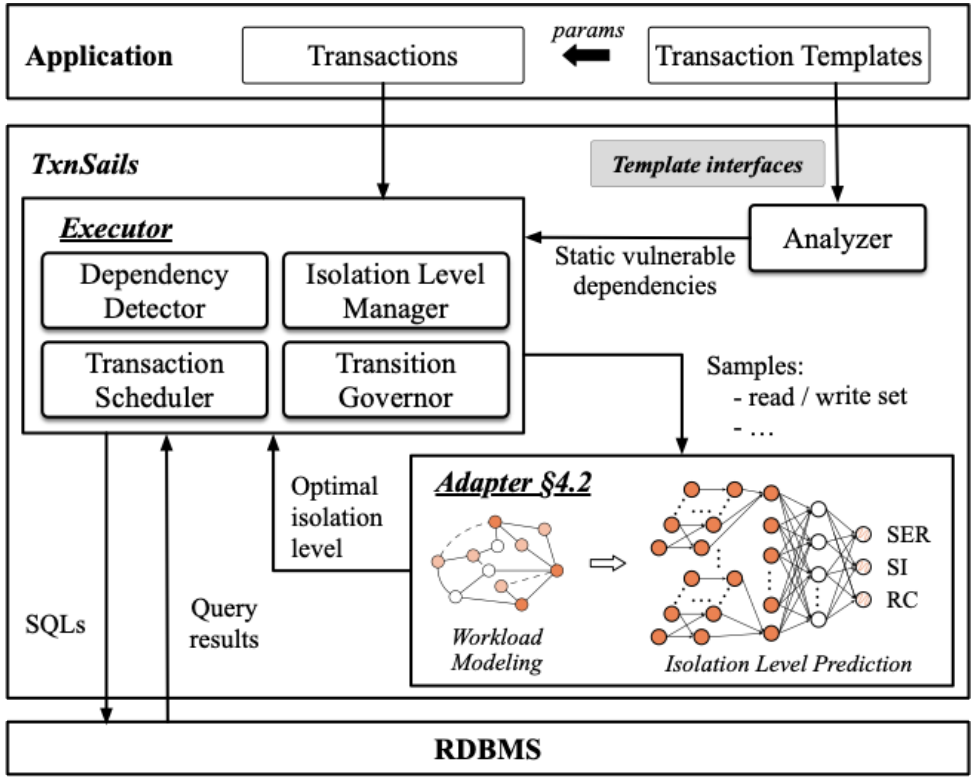
TxnSails works in the middle tier between the application and database, it comprises three main components: ***Analyzer***, ***Executor***, and ***Adapter***.

Analyzer

It builds the static dependency graph (SDG) for the transaction templates and identifies all the static vulnerable dependencies for each low isolation level.



Smallbank SDG



System Overview

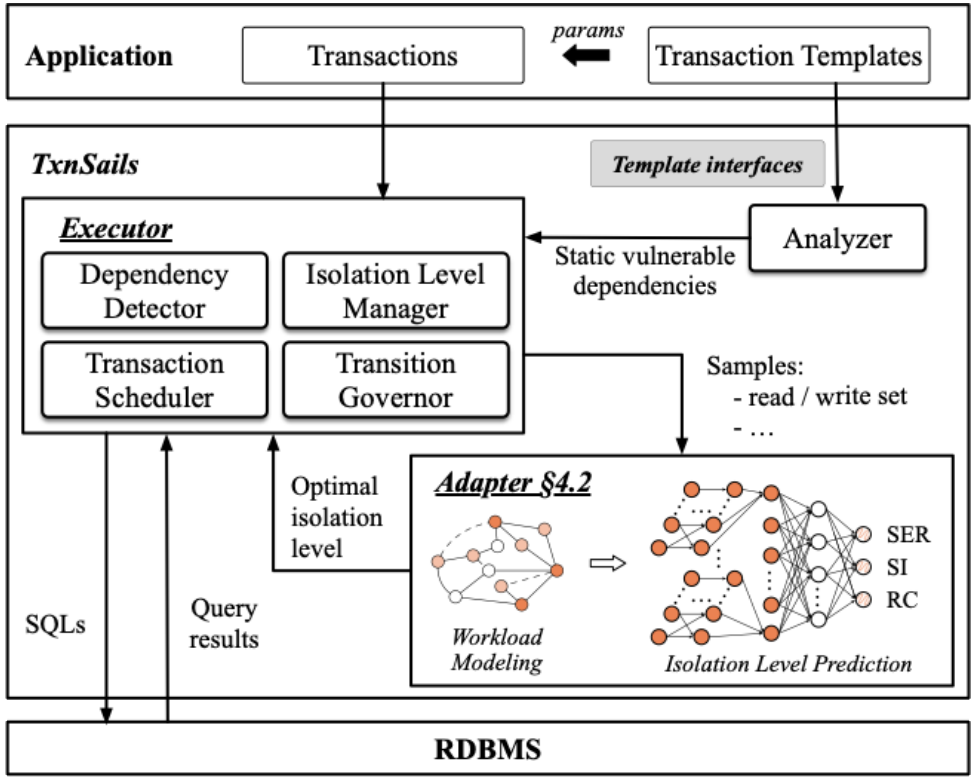
TxnSails works in the middle tier between the application and database, it comprises three main components: ***Analyzer***, ***Executor***, and ***Adapter***.

- ***Analyzer***

  It builds the static dependency graph (SDG) for the transaction templates and identifies all the static vulnerable dependencies for each low isolation level.

- ***Executor***

  It ensures SER when transactions operate at a single low isolation level or during the transition. Keep the commit order consistent with the dependency.



System Overview

TxnSails works in the middle tier between the application and database, it comprises three main components: ***Analyzer***, ***Executor***, and ***Adapter***.

- ***Analyzer***

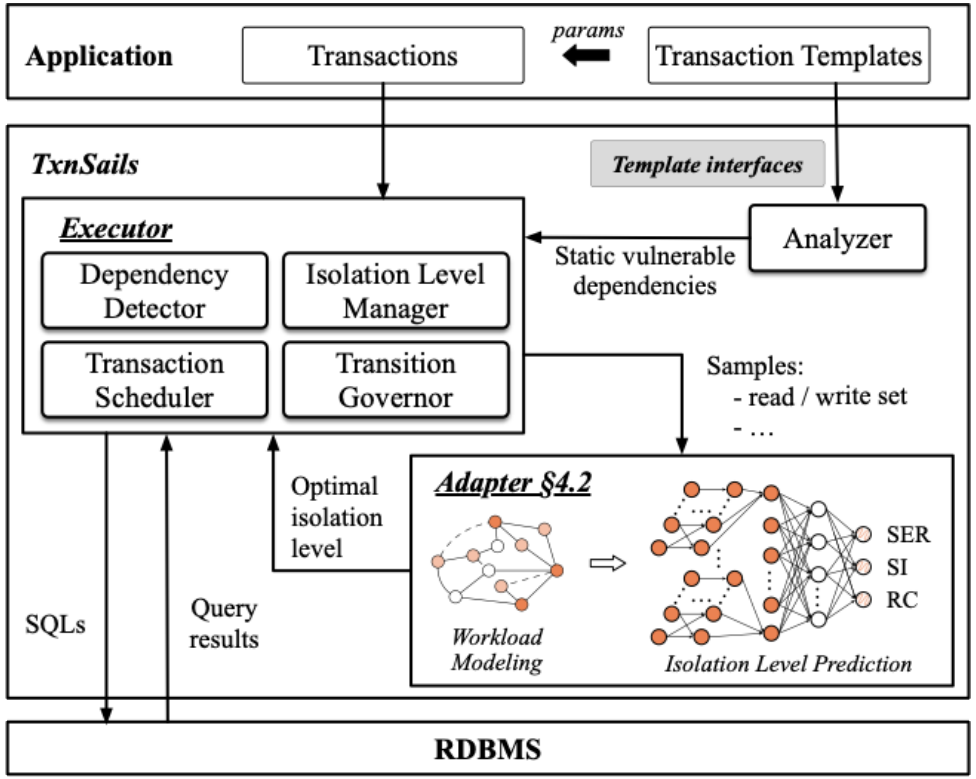
  It builds the static dependency graph (SDG) for the transaction templates and identifies all the static vulnerable dependencies for each low isolation level.

- ***Executor***

  It ensures SER when transactions operate at a single low isolation level or during the transition. Keep the commit order consistent with the dependency.

- ***Adaptor***

  It collects transactions and then employs a graph-based model to predict the optimal isolation level.



Outline

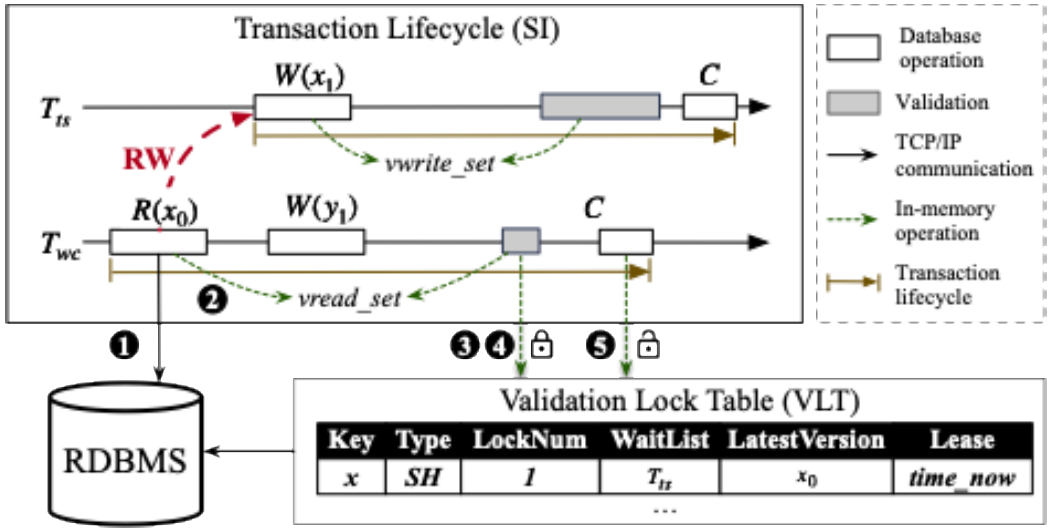
- Background
- System Overview
- **Technique Details**
- Evaluations
- Conclusion

Technique Details

Unified middle-tier concurrency control mechanism

TxnSails dynamically detects runtime vulnerable dependencies and schedules their commit order. Specifically, it adds a validation phase in transaction lifecycle before these transactions can commit.

I. In the execution phase, TxnSails stores the read/write data items in the thread-local buffer that may induce the vulnerable dependencies;
II. In the validation phase, TxnSails acquires validation locks for data items stored in the buffer. Then, it detects dependencies among them and aims to schedule the commit order consistent with the identified dependency order;
III. In the commit phase, TxnSails applies modifications to the database and subsequently releases the validation locks.



Technique Details

Self-adaptive isolation level selection

TxnSails adopts transaction dependency graphs to capture workload features and adopts a graph classification model to perform self-adaptive isolation level selection.

Graph construction

TxnSails proposes a graph-structured workload model, where each transaction is mapped to a vertex v_i, and its feature vector v_i is generated by extracting the number of data items in its read and write set. For each edge (v_i, v_j), TxnSails extracts the data dependency type and the involved relations to generate its feature.

Graph embedding and isolation level prediction

The graph model comprises two parts. First, we use a *Graph Embedding Network* to aggregate both vertex and edge features, encoding the local structure and attribute information of the graph. Second, we use a *Graph Classification Network* that learns the mapping from the embedded matrix to the optimal isolation level and perform the graph classification to predict the optimal isolation level.

Technique Details

Cross-isolation level validation mechanism

If the predicted optimal isolation level changes, TxnSails will adapt from the previous isolation level I_{old} to the optimal isolation level I_{new}. TxnSails employs a cross-isolation validation (CIV) mechanism that ensures serializability and allows for non-blocking transaction execution.

Cross-isolation vulnerable dependency

The cross-isolation vulnerable dependency is defined as $T_j \xrightarrow{rw} T_k$ in chain $T_i \xrightarrow{rw} T_j \xrightarrow{rw} T_k$, where T_j commits after the transition starts.

Corollary

For any cross-isolation vulnerable dependency $T_j \xrightarrow{rw} T_k$, if T_j ***commits before*** T_k , the scheduling achieves SER.

Technique Details

Cross-isolation level validation mechanism

If the predicted optimal isolation level changes, TxnSails will adapt from the previous isolation level I_{old} to the optimal isolation level I_{new}. TxnSails employs a cross-isolation validation (CIV) mechanism that ensures serializability and allows for non-blocking transaction execution.

Cross-isolation vulnerable dependency

The cross-isolation vulnerable dependency is defined as $T_j \xrightarrow{rw} T_k$ in chain $T_i \xrightarrow{rw} T_j \xrightarrow{rw} T_k$, where T_j commits after the transition starts.

Corollary

For any cross-isolation vulnerable dependency $T_j \xrightarrow{rw} T_k$, if T_j ***commits before*** T_k , the scheduling achieves SER.

Do middle-tier concurrency control to ensure the consistent dependency order and commit order !

Outline

- Background
- System Overview
- Technique Details
- **Evaluations**
- Conclusion

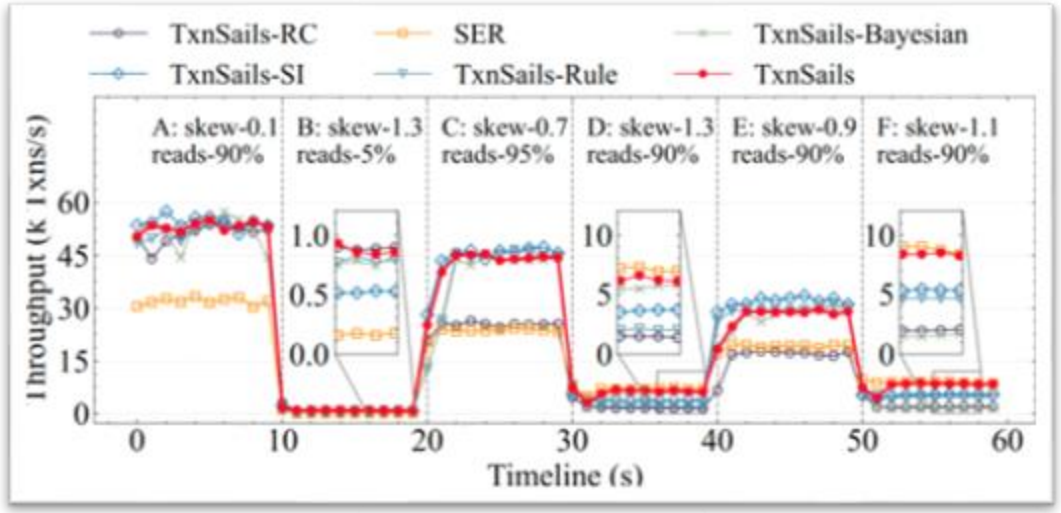
Evaluations

Self-adaptive isolation level selection

We first evaluate selfadaptive isolation level selection by varying the workload every 10s across six distinct scenarios. We sample the workload at 1-second intervals. The results demonstrate that different isolation levels perform variably under different workloads:

- ✓ SI performs well in low-skew scenarios (A,C,E).
- ✓ SER is more suitable in high-skew scenarios with little writes (D,F).
- ✓ RC excels in high skew scenarios with more writes (B).

TxnSails can choose the optimal isolation level across all tested scenarios.

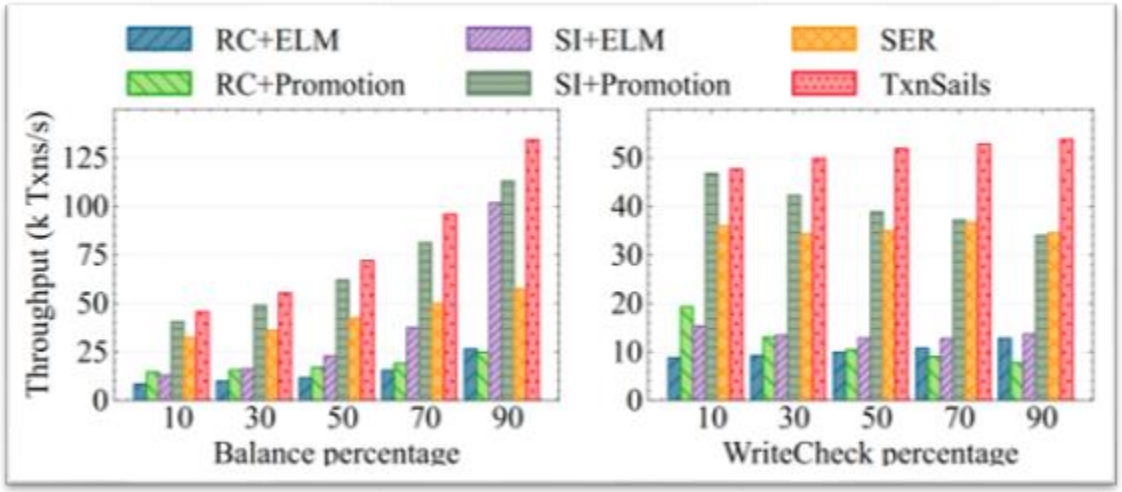


Evaluations

Impact of templates percentages

In complex workloads like SmallBank and TPC-C, only certain queries lead to data anomalies. This part compares different solutions by varying the percentage of critical transaction templates.

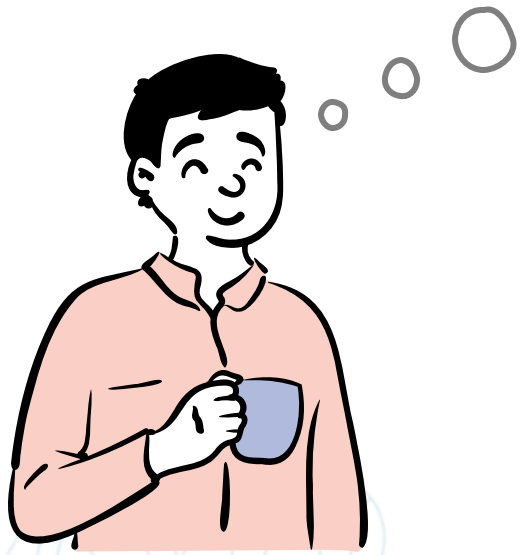
As the ratio of Balance transaction increases, TxnSails achieves up to 6.2× performance gain. As the ratio of WriteCheck transactions increases, TxnSails's advantage becomes more pronounced as no extra WW conflicts are introduced, outperforming up to 2.3× the performance of SER.



Outline

- Background
- System Overview
- Technique Details
- Evaluations
- **Conclusion**

Conclusion



I will choose TxnSails to manage database isolation level.

- TxnSails introduces a unified middle-tier validation method to enforce the commit order consistent with the vulnerable dependency order, ensuring serializability in single-isolation and cross-isolation scenarios.
- TxnSails adopts a graph learned model to extract the runtime workload characteristics and adaptively predict the optimal isolation levels, achieving further performance improvement.

Thanks!

Personal site: *https://qiyuzhuang.github.io/*

Email: *qyzhuang@ruc.edu.cn*