Simulating a Transactional Server for Multi-Model Systems

Zenon Zacouris
zenon.zacouris@tum.de
Technical University of Munich
Munich, Germany

Maribel Acosta
maribel.acosta@tum.de
Technical University of Munich
Munich, Germany

ABSTRACT

Multi-model systems integrate heterogeneous models, making consistency management a critical challenge. We present M2TS, a transactional server simulator for multi-model environments, enabling users to analyze the impact of consistency-preserving transactions on system performance. Unlike traditional transactional models that focus on ACID consistency, M2TS ensures multi-model consistency via bookkeepers, which propagate updates across models. The simulator supports various concurrency and consistency settings, allowing users to explore trade-offs in real-time. Through this demonstration, we provide insights into managing transactions in complex, interconnected environments.

PVLDB Reference Format:
Zenon Zacouris and Maribel Acosta. Simulating a Transactional Server for Multi-Model Systems. PVLDB, 18(12): 5395 - 5398, 2025.
doi:10.14778/3750601.3750680

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://purl.org/m2ts.

1 INTRODUCTION

Transactions have been extensively studied in the context of relational databases, with a strong focus on ensuring atomicity, consistency, isolation, and durability (ACID). However, modern systems increasingly incorporate multiple heterogeneous models, leading to more complex transaction scenarios.

A prominent example of such a scenario arises in the development of Cyber-Physical Systems (CPS), where multiple engineering disciplines collaborate using heterogeneous models. In CPS development, data models and models from mechanical, electrical, and software engineering must remain consistent despite frequent updates. This consistency is enforced through consistency specifications (CS), which define rules for propagating changes between interconnected models [4]. A real-world example from electric vehicle development is shown in Figure 1 (adapted from [1]).

A change in tire diameter in the CAD model must propagate to E/E and Simulink models for calibration and braking analysis.

Despite the increasing adoption of multi-model systems, the impact of consistency preservation mechanisms on transaction performance remains understudied. Existing research has focused on transactional performance in single-model systems, such as relational databases. However, in CPS and other complex system domains, transactions must handle multiple interdependent models, each with different representations and constraints.

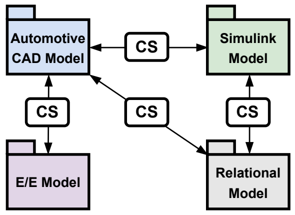


Figure 1: A multi-model system for CPS development with consistency specifications (CS)

We tackle this challenge by presenting the M2TS simulator for consistency-preserving transactions in multi-model systems. Our simulator allows users to analyze the effects of consistency specifications and multi-model interactions on transaction performance. By simulating concurrent transactions across heterogeneous models, M2TS provides insights into the main performance factors, such as transaction size, the number of processors, and the effects of consistency-preserving mechanisms. Our findings contribute to the broader understanding of transaction management in heterogeneous systems, bridging the gap between database foundations and emerging multi-model environments. The M2TS demo is available online.[1]

2 RELATED SYSTEMS AND NOVELTY

Simulators are widely used to minimize costs while offering a safe testing environment and faster feedback. In the context of transactions, existing works focus on single-schema or federated systems.

Ries et al. [6] present a single-model transaction simulator and highlight the importance of locking granularity. Locking granularity refers to the lock size used, which is crucial for balancing concurrency and performance. The authors simulated a system in equilibrium (in terms of transaction volume) to investigate the impact of locking granularity. Their findings emphasize the trade-off between fine and coarse granularity: fine granularity allows for more concurrency but increases lock overhead, while coarse granularity reduces overhead but lowers overall performance. They concluded that coarse granularity is preferred, except in scenarios where transactions randomly access small portions of the database.

Dandamudi et al. [2] extended this concept by considering multiprocessor database systems using a shared-nothing architecture. Their simulation of a distributed database system reinforced the conclusions of Ries et al. [6], showing that fine granularity is optimal for random access patterns and a small number of transactions, whereas coarse granularity is generally more efficient in other cases.

Proceedings of the VLDB Endowment, Vol. 18, No. 12 ISSN 2150-8097.
doi:10.14778/3750601.3750680

[1] M2TS Demo: https://purl.org/m2ts

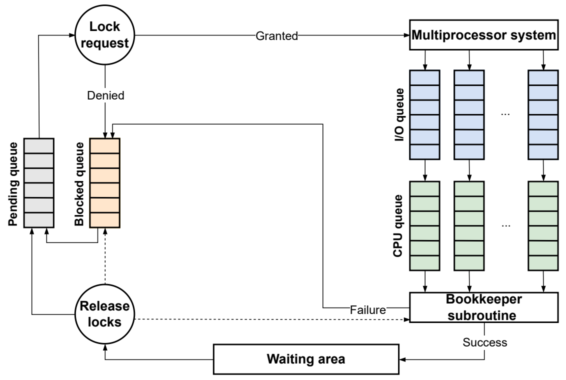


Figure 2: The M2TS main routine

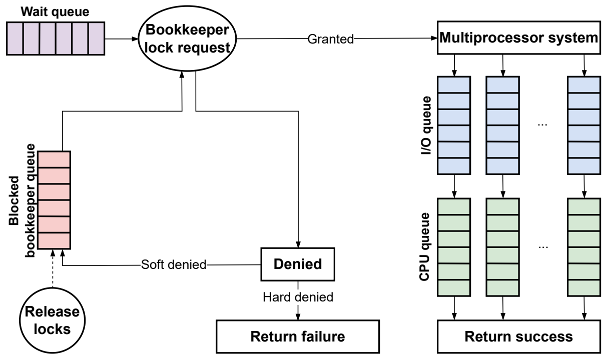


Figure 3: The M2TS bookkeeper routine

Lee et al. [5] examine how updates are propagated in federated database systems (FDBS), ensuring inter-schema consistency through careful mapping of updates across schema layers. A key distinction is that FDBS enforces strict conditions for when federated schema objects are updatable. In contrast, multi-model systems, like the one we simulate, allow updates across different models without such limitations, followed by consistency-preserving operations to maintain inter-model consistency. This demonstrates the more flexible update approach in multi-model systems compared to the rigid process in FDBS.

We deviate from existing works by simulating transactions across multiple models within a unified system. Unlike federated databases, where each component is autonomous and update propagation cannot be part of an atomic transaction, our system assumes central control. This allows consistency-preserving updates to be executed transactionally, ensuring cross-model consistency without requiring external coordination.

3 OUR APPROACH: A MULTI-MODEL TRANSACTION SIMULATOR (M2TS)

The goal of our work is to study the effects of various settings on transactional servers in multi-model environments, providing researchers and practitioners with a flexible tool to understand the performance of transactions under different scenarios. For this, M2TS consists of the server simulator (§3.1) implemented in Java SE-17, and the user interface (§3.2) to set up the simulator parameters and visualize the results, which is implemented in Vue.js 5.0.8. The user interface and the server simulator are integrated via an API developed using Spring Boot 3.3.4.

3.1 Transactional server simulator

In this work, we devise a transaction server that ensures consistency. For this, the system employs two complementary mechanisms.

ACID Isolation and Multi-Model Consistency. Isolation in ACID ensures that transactions execute independently, preventing anomalies such as *dirty reads, non-repeatable reads, and phantom reads.* Our system enforces **strong strict two-phase locking (SS2PL)**, which guarantees *serializability*, the strongest form of isolation. This ensures that transactions execute as if they were serialized, eliminating interference between concurrent transactions.

ACID consistency ensures that a transaction transitions the database from one valid state to another by enforcing predefined *integrity constraints*. However, in multi-model environments, additional mechanisms are required to ensure consistency across heterogeneous models. To address this, M2TS employs **bookkeepers**, dynamic consistency-preserving transactions that propagate updates across models according to predefined *consistency specifications*. Since bookkeepers execute alongside primary transactions, they introduce challenges related to lock management and concurrency control. To prevent deadlocks under SS2PL, our system follows a *cautious waiting protocol*. If a bookkeeper cannot acquire a lock, the system checks whether the blocking transaction is itself waiting. If it is, the parent transaction and all its bookkeepers are reverted; otherwise, the bookkeeper is placed in the blocked bookkeeper queue. This ensures that multi-model consistency is maintained efficiently while avoiding deadlocks.

Together, the locking mechanism and bookkeepers, implemented via specialized queues in the main and bookkeeper routines, ensure the overall consistency of the system.

Queues. The system relies on several types of queues:

- **Pending queue**: Initially, transactions are placed in this queue. They await processing and are moved based on lock acquisition.
- **Blocked transaction queue**: Transactions that cannot acquire necessary locks are moved here until they are unblocked.
- **Wait queue**: If updates trigger the creation of bookkeepers, these are placed in a wait queue, ensuring they are processed in the correct order.
- **Blocked bookkeeper queue**: If a bookkeeper is soft-denied a lock it is placed in a blocked bookkeeper queue.
- **I/O queue**: Once transactions are ready to proceed, they are moved to the I/O queue. Transactions are distributed across multiple I/O queues to balance the load.
- **CPU queue**: After I/O operations, transactions are processed in the CPU queue, where computational tasks are performed.

These queues ensure efficient and deadlock-free processing by prioritizing bookkeepers and ensuring synchronization across models.

Main routine. The simulator starts with initializing the transactions and models. As shown in Figure 2, the transactions are put in a pending queue and try to obtain locks one by one. If granted a lock, the transaction is passed on to one of the *n* I/O queues – where

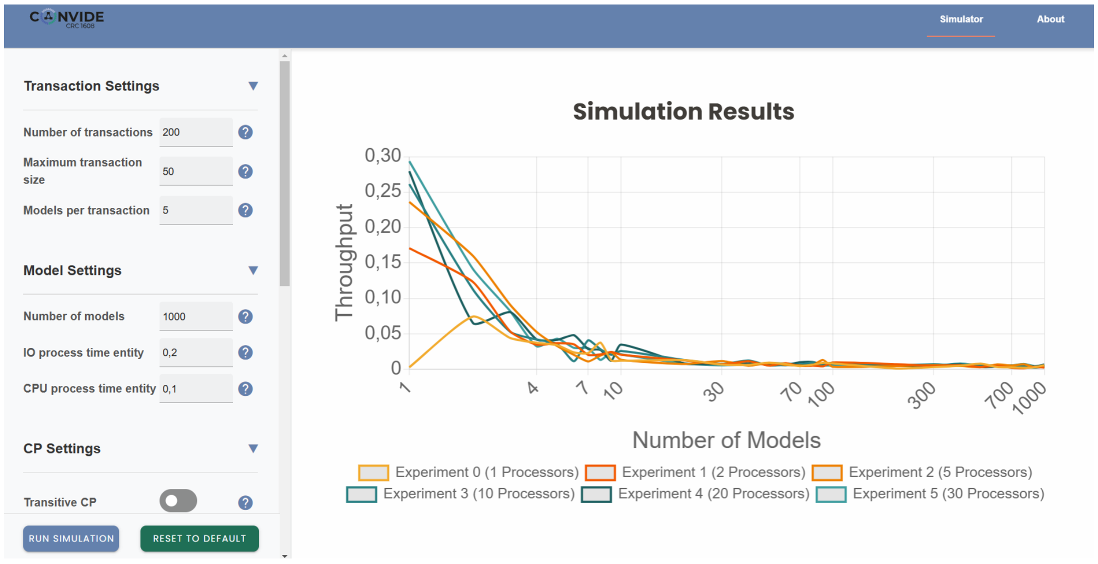

CONVIDE
CRC 1608
Simulator
About
Transaction Settings
Number of transactions 200
Maximum transaction size 50
Models per transaction 5
Model Settings
Number of models 1000
IO process time entity 0,2
CPU process time entity 0,1
CP Settings
Transitive CP
RUN SIMULATION
RESET TO DEFAULT
Simulation Results
Throughput
0,30
0,25
0,20
0,15
0,10
0,05
0
1
4
7
10
30
70
100
300
700
1000
Number of Models
Experiment 0 (1 Processors)
Experiment 1 (2 Processors)
Experiment 2 (5 Processors)
Experiment 3 (10 Processors)
Experiment 4 (20 Processors)
Experiment 5 (30 Processors)

Figure 4: The M2TS user interface. The left panel includes the simulator parameters. The simulation results are plotted in the main panel, showing the throughput (fraction of transactions executed per unit time) w.r.t. the number of models in the system.

n is the number of processors. If they cannot be granted a lock, the transaction is appended to the blocked queue, where they remain until the transaction blocking them has finished execution. This mimics the SS2PL protocol, where a transaction does not acquire any further locks after it has released its first lock. To achieve this, we enforce that a transaction needs to have all necessary locks before starting execution. The transactions in the I/O queue are processed in parallel. After processing, they are put into one of the n CPU queues, and once they are finished, potential bookkeepers are started to ensure consistency after all updates. If no bookkeepers are started, the transaction automatically finishes, and a new one will start. If bookkeepers have been generated, the so-called bookkeeper routine starts.

Bookkeeper routine. Figure 3 shows that the bookkeepers follow the same processing steps as transactions; they first try to get locks and then process the locked entities. A novelty compared to the main routine is the locking protocol: In the main routine, we use a SS2PL protocol. Bookkeepers are dynamically started "sub-transactions" that must be in the same atomic unit as the parent transaction. When a bookkeeper tries to get a lock, and it fails, one of two cases can occur: (1) *soft denial:* the blocking transaction is not waiting, so the bookkeeper is put in the waiting queue, or (2) *hard denial:* the blocking transaction is waiting, and the bookkeeper, its parent transactions, and all the other bookkeepers belonging to it are reverted and restarted. Once a bookkeeper has finished execution, it signals to its parent transaction that it is done. Once all bookkeepers of a transaction have finished executing, the transaction finishes and a new transaction is started.

3.2 User interface and simulator parameters

M2TS includes a user interface (Figure 4), that enables the configuration of simulator parameters and the simulation results. Since multi-model transactions are a key novelty of our work, the simulation results show the impact of different model counts on throughput. The simulator parameters include:

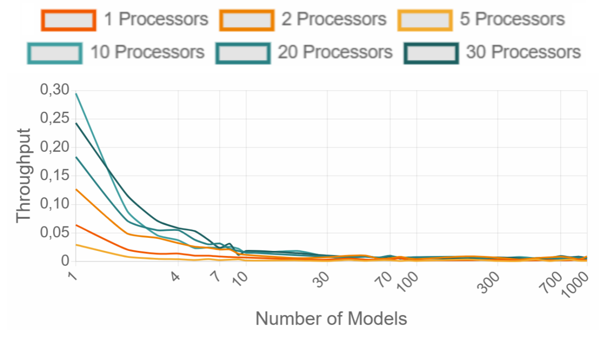
Transaction settings. These include the number of transactions simultaneously executed in the simulation. The system is in an equilibrium state, keeping the total number of transactions constant. The maximum transaction size can be set, limiting the uniform distribution sampled to get the transaction size for a particular transaction. The user can also set the number of models a transaction affects to simulate multi-model environments.

Model settings. The user can set the number of models the system consists of, which is also the upper bound of the number of models per transaction. Industrial MBSE projects can involve over one hundred interconnected models distributed across teams and tools [3]. These parameters also include the I/O and CPU time to execute updates over an element of a model.

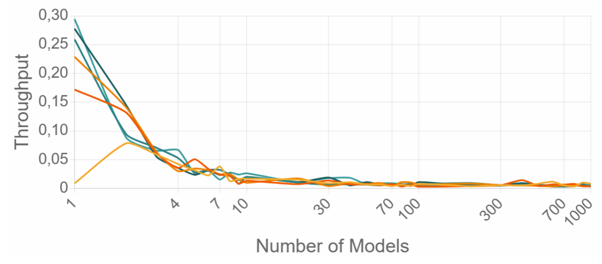
Consistency preservation settings. The user can change the number of bookkeepers started during a simulation run. The user can select if bookkeepers can trigger other bookkeepers (*transitive CP*). Next, the maximum number of bookkeepers triggered by a single transaction can be set - this can be varied to simulate the effects of many bookkeepers on the system performance.

Lock settings. The number of locks determines what portion of a model is covered by each lock. If we have 5000 elements in a model and 200 locks, then every lock covers 25 elements. In short, this determines the lock granularity. Next, the user can set the I/O and CPU time to lock a single element.

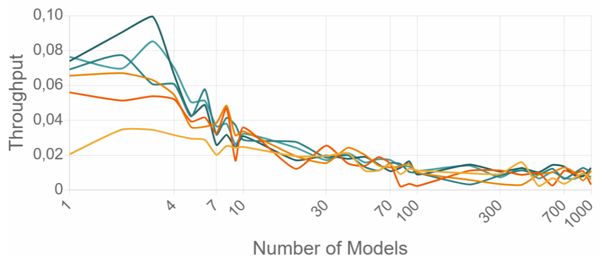
System settings. The system size defines the number of elements in the system. The number of steps taken in the simulation can also be varied. Lastly, the number of processors allows the user to look at the impact of dividing the work over more cores.



(a) High Latency



(b) In-Memory Database



(c) Heavy Load

Figure 5: Three distinct scenarios illustrating system behavior, with the x-axis displayed on a logarithmic scale.

4 DEMONSTRATION

In this section, we describe how users can interact with the M2TS simulator (see Figure 4). This will be the first public demonstration of M2TS. Attendees will have the opportunity to interact with the system, configure simulation parameters, and analyze multi-model transaction performance live. We report throughput as it reflects the system's ability to process transactions under concurrency and consistency constraints, and serves as a key indicator of how well model synchronization scales under load.

The number of models and the number of processors are varied over the experiments and are hence indicated with variable names n and m. For this demonstration, we provide in the online tool a predefined set of parameter values that allow users for exploring different system behaviors, such as *high-latency environments*, *in-memory storage*, and *heavy workloads*.

High-latency system. For this scenario, we use the default parameters from the M2TS Demo[1] and increase the I/O lock and processing time (*IOLockTime*, *IOProcessTime* = 10). We analyze how multi-model transactions behave under varying processor and model counts. Results in Figure 5a show that at low processor counts, latency severely impacts throughput, while at higher processor counts, parallelism offsets some of the delay. However, as model count increases, performance declines rapidly due to the overhead of spawning multiple bookkeepers to maintain consistency. Figure 5a shows that processor count does not positively correlate with throughput, confirming that locking time, not processing time, is the main bottleneck.

System with in-memory storage. To analyze a low-latency system, we take the same default parameters and set *IOLockTime* and *IOProcessTime* to 0, simulating an in-memory database. Results indicate that for a single-model system, throughput is higher for all processor counts except a single processor. However, as model count increases, performance deteriorates similarly to the high-latency case. Figure 5b confirms that in both scenarios, the bottleneck remains lock acquisition, not processing time.

System under heavy load. To simulate a system flooded with many small transactions, we use the same default parameters and increase the *numberOfTransactions* to 1000. As expected, throughput drops significantly due to higher lock contention. However, at higher model counts, throughput converges to a similar degraded state as in previous scenarios, reinforcing that lock contention dominates in multi-model transaction performance.

4.1 User Interaction Walkthrough

Attendees interact with M2TS through a web interface offering configurable parameters and predefined presets. Each parameter has bounded input ranges with fixed steps. After configuration or preset selection, clicking Run Simulation executes the scenario. The output visualizes throughput for 1, 2, 5, 10, 20, and 30 processors, reflecting system performance impacts.

Across all three scenarios, we observe that as the number of models increases, throughput rapidly degrades toward zero. This makes the performance impact of concurrent transactions in multi-model systems explicit, motivating future work on relaxing ACID constraints to ensure scalability.

ACKNOWLEDGMENTS

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SFB 1608 – 501798263.

REFERENCES

[1] Maribel Acosta, Sebastian Hahner, Anne Koziolek, Thomas Kühn, Raffaela Mirandola, and Ralf Reussner. 2022. Uncertainty in coupled models of cyber-physical systems. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 569–578.

[2] S Dandamudi and S-L Au. 1991. Locking granularity in multiprocessor database systems. In *Proceedings of the Seventh International Conference on Data Engineering*. IEEE, 268–269.

[3] Jeff A Estefan. 2007. Survey of model-based systems engineering (MBSE) methodologies. *INCOSE MBSE Initiative* 25, 8 (2007), 1–12.

[4] Heiko Klare, Max E Kramer, Michael Langhammer, Dominik Werle, Erik Burger, and Ralf Reussner. 2021. Enabling consistency in view-based system development—the vitruvius approach. *Journal of Systems and Software* 171 (2021), 110815.

[5] Mong Li Lee, Sin Yeung Lee, and Tok Wang Ling. 2001. Updatability in federated database systems. In *Proceedings of the 12th International Conference on Database and Expert Systems Applications*. Springer, 2–11.

[6] Daniel R Ries and Michael R Stonebraker. 1979. Locking granularity revisited. *ACM Transactions on Database Systems* 4, 2 (1979), 210–227.