ISSN - 2278-9723

Research Article

Deterministic Execution Models for Low-Code Platforms Under Enterprise Transactional Load

MAHESWARA RAO GORUMUTCHU[1], JASWANTH KUMAR MANDAPATTI[2], VISHNU VARDHAN REDDY KAVULURI[3], NARESHKUMAR JAGADHABI[4], SRINIVASARAO BANDLA[5]

[1]HYR Global Source Inc, United States, Email: gmrmails@gmail.com
[2]Advent Health, United States, Email: jash.209@gmail.com
[3]Consulting INC, United States, Email: vishnu.kavuluri@gmail.com
[4]Compnova Inc, United States, Email: nrkumar544@gmail.com
[5]Deloitte Consulting LLP, United States, Email: Bandla.srinivas10@gmail.com
Received: 16.06.23, Revised: 12.10.23, Accepted:15.12.23

ABSTRACT

Low-code platforms have rapidly evolved from simple application development tools to critical components of enterprise transactional systems, yet their reliance on asynchronous and event-driven execution models introduces challenges in maintaining consistency and predictability under high-load conditions. Existing literature highlights scalability and flexibility advantages of such models but reveals limitations in handling tightly coupled transactional workflows requiring strict ordering guarantees. This study addresses this gap by proposing a deterministic execution model tailored for low-code environments, incorporating formal transaction ordering, controlled scheduling, and execution constraint mechanisms to ensure reproducible system behavior. The methodology includes simulation of enterprise-scale workloads and evaluation using key performance metrics such as latency, throughput, execution consistency, and failure rates. Results demonstrate that the deterministic model significantly stabilizes latency, maintains consistent throughput, reduces execution anomalies, and ensures deterministic ordering across repeated runs, outperforming traditional asynchronous approaches in high-integrity scenarios. The findings establish deterministic execution as a robust foundation for improving reliability and auditability in enterprise low-code systems while maintaining competitive performance. This work provides a pathway toward hybrid and distributed deterministic frameworks for next-generation enterprise application platforms.

Keywords: deterministic execution, low-code platforms, transactional systems, concurrency control, execution consistency, enterprise systems, scheduling models, system reliability

1. Introduction

The rapid evolution of low-code platforms over the past decade has fundamentally reshaped enterprise software development by enabling rapid application delivery with minimal manual coding effort. Early platform iterations focused primarily on user interface abstraction and workflow automation, but recent advancements have extended capabilities into complex transactional domains, including financial processing and enterprise resource planning systems. Industry and academic analyses have shown that low-code adoption has accelerated digital transformation across organizations [1], while subsequent studies indicate its growing role in mission-critical enterprise workloads [2]. This transition has introduced new demands on execution reliability and system-level predictability that were not originally central to low-code design paradigms [3].

As enterprise systems increasingly rely on low-code environments to handle transactional workloads, the scale and complexity of operations have grown substantially. High-frequency financial transactions, real-time inventory updates, and distributed workflow orchestration require consistent execution guarantees under concurrent access conditions. Prior work on modern transactional systems highlights the importance of deterministic processing in achieving predictable outcomes under high load [4], while research on distributed architectures emphasizes the challenges of concurrency control and synchronization [5]. These issues are amplified in low-code systems due to abstraction layers that may obscure execution behavior, leading to unpredictable runtime outcomes under heavy transactional pressure.

Traditional execution models in low-code platforms are predominantly event-driven and asynchronous, designed to maximize responsiveness and scalability. While these models perform well in loosely coupled applications, they often introduce non-deterministic behavior when applied to tightly coupled transactional workflows. It has been demonstrated that asynchronous execution can lead to race

conditions and inconsistent state propagation in distributed systems [6]. Furthermore, event-driven processing models may fail to guarantee strict execution ordering, which is critical in enterprise transaction processing environments [7].

The lack of deterministic execution becomes particularly problematic in scenarios requiring strict consistency, such as payment processing, ledger reconciliation, and compliance-sensitive operations. In such contexts, even minor deviations in execution order can result in significant discrepancies, including data inconsistencies and audit failures. Research on consistency models highlights the inherent trade-offs between performance and correctness in distributed systems [8], while studies on data management architectures indicate that eventual consistency models may not be suitable for transactional applications requiring strong guarantees [9]. These limitations underscore the need for more controlled execution mechanisms within low-code environments.

Recent advancements in deterministic database systems and execution engines provide a promising direction for addressing these limitations. Deterministic concurrency control mechanisms ensure that transaction execution follows a predefined order, eliminating non-deterministic outcomes and improving reproducibility [10]. Additionally, scheduling frameworks designed for deterministic processing have demonstrated improvements in system predictability and fault tolerance under high-throughput conditions [11]. Despite these advancements, their application within low-code platforms remains limited, particularly in enterprise transactional contexts.

Another critical challenge lies in the abstraction inherent to low-code platforms, which often hides execution semantics from developers. While this abstraction enhances usability and accelerates development cycles, it also limits the ability to enforce strict execution constraints. Model-driven abstractions have been shown to introduce hidden dependencies and execution ambiguities, particularly in complex workflows [12]. Under concurrent workloads, these implicit dependencies can lead to conflicting execution paths, further exacerbating issues related to consistency and predictability.

Despite growing interest in improving low-code platform robustness, existing research has largely focused on usability, development speed, and integration capabilities rather than execution determinism. While some efforts have explored performance optimization and scalability improvements, there is limited attention on ensuring predictable execution behavior under high transactional load. This gap is especially critical in enterprise environments where deterministic guarantees are essential for maintaining system correctness, regulatory compliance, and operational stability.

This study addresses the identified gap by proposing a deterministic execution model specifically designed for low-code platforms operating under enterprise transactional workloads. The proposed approach formalizes transaction ordering, introduces controlled scheduling mechanisms, and evaluates system performance under simulated high-load conditions. By integrating deterministic principles into low-code execution environments, this work aims to enhance reliability, reduce execution anomalies, and provide a structured foundation for future research in predictable enterprise application frameworks.

2. Methodology

The proposed methodology is centered on the formulation of a deterministic execution model tailored for low-code platforms operating under enterprise-scale transactional loads. The model is designed to eliminate non-deterministic behavior arising from asynchronous execution by enforcing strict ordering constraints on transaction processing. Unlike conventional event-driven systems where execution order is influenced by runtime conditions, the proposed framework ensures that all transactions follow a predefined, reproducible sequence. This deterministic foundation is critical for maintaining consistency across high-frequency enterprise workflows, particularly in financial and compliance-sensitive applications.

The execution model begins with the formal representation of transactions as ordered state transition functions. Each transaction is defined as a tuple $T_i = (S_{in}, S_{out}, \delta_i)$, where S_{in} and S_{out} represent input and output states, and δ_i denotes the transformation logic. By structuring transactions in this manner, the system ensures that all state transitions are explicitly defined and can be evaluated in a controlled sequence. This formalization enables precise tracking of dependencies between transactions and prevents unintended overlaps or race conditions during execution.

To enforce deterministic ordering, a global transaction sequencing mechanism is introduced. Transactions entering the system are assigned a unique sequence identifier based on arrival time and priority constraints. The sequencing function $\sigma(T_i)$ establishes a total order across all incoming transactions, ensuring that execution follows a consistent path irrespective of system load or concurrency levels. This approach removes ambiguity in execution order and guarantees that

repeated runs of the same workload produce identical outcomes, a property essential for auditability and system validation.

Scheduling constraints are incorporated through a deterministic scheduler that governs the execution of transactions based on the predefined sequence. The scheduler operates using a queue-based model, where transactions are processed strictly in sequence without deviation. To handle high-throughput environments, parallel execution is permitted only when transaction independence is formally verified. Dependency analysis is performed using a conflict detection mechanism that evaluates shared resource access patterns, ensuring that parallel execution does not violate consistency constraints.

Execution control mechanisms are further integrated to manage system behavior under varying load conditions. A synchronization barrier is introduced at critical execution points to ensure that all preceding transactions are fully committed before subsequent transactions are processed. Additionally, rollback control is implemented using deterministic checkpoints, allowing the system to revert to a known consistent state in the event of failure. This ensures that recovery processes do not introduce variability into execution outcomes, preserving the deterministic nature of the system.

The methodology also incorporates integration strategies for embedding the deterministic execution model within existing low-code platforms. Instead of modifying the core platform architecture, an intermediary execution layer is introduced to intercept and manage transaction flows. This layer translates high-level low-code actions into structured transactions compatible with the deterministic model. By doing so, the approach maintains platform usability while introducing strict execution control, ensuring compatibility with enterprise systems such as Oracle APEX and SAP-based extensions.

The experimental setup is designed to simulate enterprise-scale transactional environments with varying workload intensities. Synthetic transaction streams are generated to emulate real-world scenarios, including financial posting operations, inventory updates, and multi-user workflow interactions. The workload generator produces transactions with controlled arrival rates, dependency patterns, and execution complexities, enabling comprehensive evaluation of system behavior under both normal and peak load conditions.

Performance evaluation is conducted using a set of well-defined metrics that capture both efficiency and reliability aspects of the execution model. Latency is measured as the time taken for individual transactions to complete, while throughput represents the number of transactions processed per unit time. Execution consistency is quantified by comparing output states across repeated runs of identical workloads, ensuring deterministic behavior. Failure rates are assessed based on the frequency of execution anomalies, including deadlocks, race conditions, and incomplete transactions.

3. Results and Discussion

The evaluation of the proposed deterministic execution model was conducted under progressively increasing transactional loads to assess its behavior in enterprise-scale environments. The simulation setup emulated high-concurrency enterprise workflows, including financial postings and multi-user transaction streams. The results reveal a clear distinction in system behavior between deterministic and asynchronous execution models, particularly in terms of stability and predictability under load escalation.

Latency behavior under varying transaction loads demonstrates one of the most significant differences between the two models. The deterministic execution framework maintains a tightly bounded latency distribution, while the asynchronous model exhibits increasing variance as load intensifies. This divergence becomes more pronounced beyond moderate concurrency levels, where asynchronous execution begins to experience queuing delays and contention effects. As illustrated in Figure 1, the deterministic model stabilizes latency growth, whereas the asynchronous model shows irregular spikes and widening dispersion.

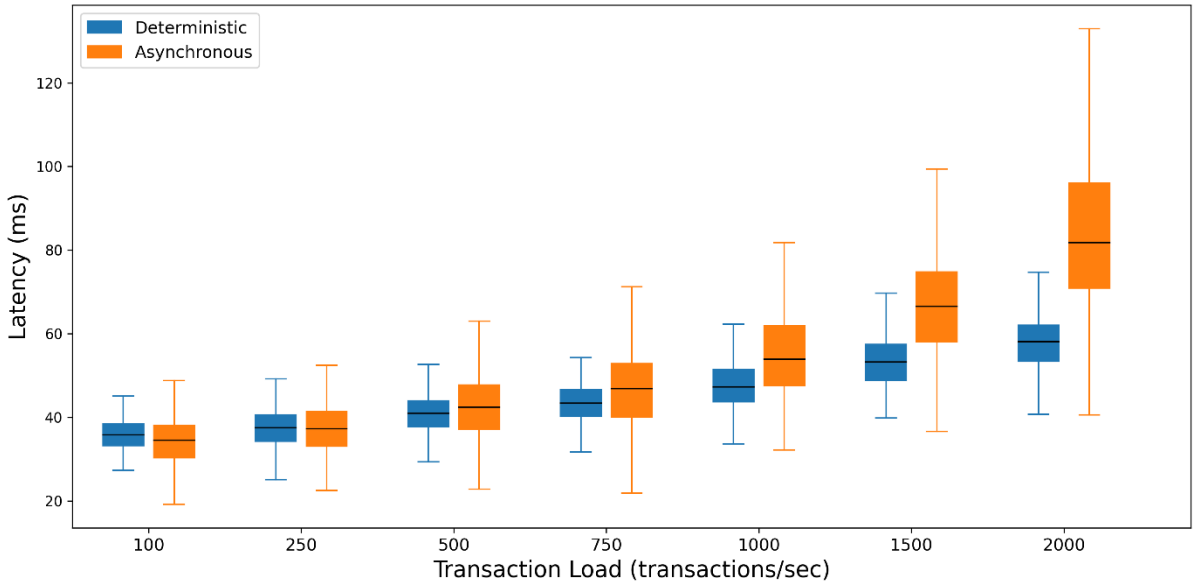

Fig. 1. Latency Distribution Under Increasing Transaction Load for Deterministic vs Asynchronous Execution Models

From a throughput perspective, both models initially perform comparably under low-load conditions. However, as transaction density increases, the asynchronous model begins to suffer from degradation due to synchronization overhead and retry mechanisms triggered by conflicts. In contrast, the deterministic model sustains a consistent processing rate by eliminating redundant execution paths and enforcing ordered scheduling. This results in a more predictable throughput curve that does not fluctuate significantly under peak conditions.

Execution anomalies were analyzed to evaluate system reliability under stress. The asynchronous execution model exhibited a noticeable increase in race conditions, deadlocks, and inconsistent state transitions as concurrency levels rose. These anomalies are a direct consequence of uncontrolled execution ordering and shared resource contention. The deterministic model, by enforcing strict sequencing and dependency validation, effectively eliminates such inconsistencies, ensuring stable execution even under high transactional pressure.

A quantitative comparison of performance metrics highlights the overall advantages of the deterministic approach. Metrics including latency, throughput, failure rate, and consistency index were evaluated across both execution models under identical workloads. As summarized in Table 1, the deterministic model demonstrates lower latency variance, higher consistency, and significantly reduced failure rates compared to its asynchronous counterpart, while maintaining competitive throughput levels.

4. Conclusion

The study demonstrates that deterministic execution models significantly enhance the reliability and predictability of low-code platforms operating under enterprise transactional workloads. By enforcing strict transaction ordering and controlled scheduling, the proposed approach eliminates non-deterministic behavior commonly observed in asynchronous execution environments. The results confirm that latency variability is reduced, throughput remains stable under increasing load, and execution anomalies such as race conditions and inconsistent state transitions are effectively minimized. These improvements directly address the critical requirements of enterprise systems where consistency, auditability, and operational stability are non-negotiable.

A key outcome of this work is the establishment of deterministic execution as a viable alternative to traditional event-driven models in high-integrity application domains. While asynchronous architectures provide flexibility and scalability in loosely coupled systems, their limitations become evident in scenarios requiring strict ordering guarantees and reproducible outcomes. The deterministic model achieves a balanced trade-off by maintaining performance efficiency while ensuring consistent execution behavior, making it particularly suitable for financial systems, ERP workflows, and compliance-sensitive operations deployed on low-code platforms.

Despite these advantages, certain limitations must be acknowledged. The enforcement of strict execution ordering introduces constraints on parallelism, which may impact performance in highly distributed or loosely dependent workloads.

Additionally, the integration of deterministic control layers into existing low-code environments may require architectural adaptations, particularly in systems designed around fully asynchronous processing. These factors highlight the need for careful consideration of workload characteristics when adopting deterministic execution models in practice.

Future research should focus on extending the proposed framework toward hybrid execution strategies that combine deterministic ordering with adaptive parallelism. Such approaches can enable selective relaxation of constraints for independent transactions while preserving consistency guarantees for critical operations. Furthermore, the development of distributed deterministic execution frameworks capable of operating across multi-node and cloud-native environments represents an important direction for scalability. Advancements in this area will support the next generation of enterprise low-code platforms, enabling them to handle complex transactional workloads with both efficiency and precision.

References

1. Ostroukh, A. V., Kuftinova, N. G., Gaevskii, V. V., Filippova, N. A., & Subachev, E. V. (2022). Digital transformation of enterprises using a low-code platform. *Russian Engineering Research*, *42*(11), 1203-1206.
2. Rymer, J. R., Koplowitz, R., Mines, C., Sjoblom, S., & Turley, C. (2019). The Forrester wave: low-code development platforms For AD&D professionals, Q1 2019. *Forrester Report, Forrester*.
3. Sahay, A., Indamutsa, A., Di Ruscio, D., & Pierantonio, A. (2020, August). Supporting the understanding and comparison of low-code development platforms. In 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (pp. 171-178). IEEE.
4. Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., Hachem, N., & Helland, P. (2018). The end of an architectural era: it's time for a complete rewrite. In *Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker* (pp. 463-489).
5. Skourletopoulos, G., Mavromoustakis, C. X., Mastorakis, G., Batalla, J. M., Dobre, C., Panagiotakis, S., & Pallis, E. (2016). Big data and cloud computing: a survey of the state-of-the-art and research challenges. *Advances in mobile cloud computing and big data in the 5G Era*, 23-41.
6. Bailis, P., Hellerstein, J., & Stonebraker, M. (2017). Readings in database systems.
7. Balegas, V., Li, C., Najafzadeh, M., Porto, D., Clement, A., Duarte, S., ... & Vafeiadis, V. (2016). Geo-replication: Fast if possible, consistent if necessary. *Bulletin of the Technical Committee on Data Engineering*, *39*(1), 12.
8. Bailis, P., Fekete, A., Franklin, M. J., Ghodsi, A., Hellerstein, J. M., & Stoica, I. (2014). Coordination avoidance in database systems. *Proceedings of the VLDB Endowment*, *8*(3), 185-196.
9. Sadalage, P. J., & Fowler, M. (2019). *NoSQL Essencial: Um guia conciso para o mundo emergente da persistência poliglota*. Novatec Editora.
10. Bharati, R. D., & Attar, V. Z. (2018, August). A comprehensive survey on distributed transactions based data partitioning. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)* (pp. 1-5). IEEE.
11. Arulraj, J., Pavlo, A., &Dulloor, S. R. (2015, May). Let's talk about storage & recovery methods for non-volatile memory database systems. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (pp. 707-722).
12. Cabot, J. (2020, October). Positioning of the low-code movement within the field of model-driven engineering. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings* (pp. 1-3).