European Journal of AI, Computing & Informatics

Vol. 1 No. 3 2025

Article **Open Access**

Data Consistency and Performance Scalability Design in High-Concurrency Payment Systems

Yue Qi [1,*]

[1] Monetization, Meta Platforms, Inc., Washington, 98005, USA
* Correspondence: Yue Qi, Monetization, Meta Platforms, Inc., Washington, 98005, USA

Abstract: In the blowout development of mobile payment and Internet financial payment systems, highly concurrent payment systems face dual challenges of data consistency assurance and system performance expansion. To address these issues, design strategies including hierarchical consistency control, data distribution optimization, and consistency scalability collaborative regulation are proposed to achieve a system architecture that balances transaction accuracy and throughput. The analysis of three typical cases-aggregated payment, high-speed sales system, and financial remittance channel-demonstrates that this strategy can be applied to complex operation scenarios, enhancing efficiency and providing a theoretical foundation and practical guidelines for future development of high-reliability payment systems.

Keywords: high concurrency payment system; data consistency; performance scalability; distributed architecture

Received: 18 August 2025
Revised: 03 September 2025
Accepted: 24 September 2025
Published: 05 October 2025

1. Introduction

With the rapid development of mobile payments, e-commerce, and financial digitization, payment systems are facing huge concurrent processing demands. However, in a distributed architecture environment, there is a natural tension between data consistency and system scalability: strong consistency may lead to performance degradation, and excessive pursuit of performance may cause transaction confusion and data conflicts. The main challenge is how to ensure the security and consistency of transactions while also having flexibility, scalability, and high response speed. This article combines the characteristics of the system to provide multi-level consistency and scalability solutions. Through examples in important scenarios, it provides theoretical support for the architecture optimization and design application of highly reliable payment systems.

2. Operational Characteristics of High-Concurrency Payment Systems

2.1. Distributed Evolution Characteristics of System Architecture

As the concurrent pressure of payment business continues to rise, traditional structures cannot meet the requirements of system stability, scalability, throughput, and other aspects [1]. The old-fashioned payment system integrates core functions such as transaction execution, accounting, and risk control verification into different programs, and conducts all business operations through a single database and access portal. Although this mode is easy to deploy, it can create efficiency bottlenecks under heavy load conditions: tightly coupled modules and single-point components are prone to failure and cannot scale horizontally, becoming the main bottleneck of system reliability.

To improve the system's resilience and ability to handle parallel loads, payment systems have evolved into a decentralized collection of microservices. The main functions are divided into independent microservice components deployed to node clusters, and these nodes are managed uniformly. Asynchronous and message-driven concurrent collaboration modes are adopted between services, and Kafka is used for state migration and serial decoupling. For data storage, a sharding approach is adopted to improve read and write efficiency and scalability. This architecture pattern supports fault isolation, dispatching, scheduling, and other functions, thereby strengthening scalability and data consistency in later stages.

As shown in Figure 1, the typical architecture of a high-concurrency payment system consists of a user access layer, a service gateway layer, a core service layer, a cache layer, and a data persistence layer. This architecture achieves mutual decoupling based on service routing, which also provides a foundation for subsequent data consistency and performance scalability.

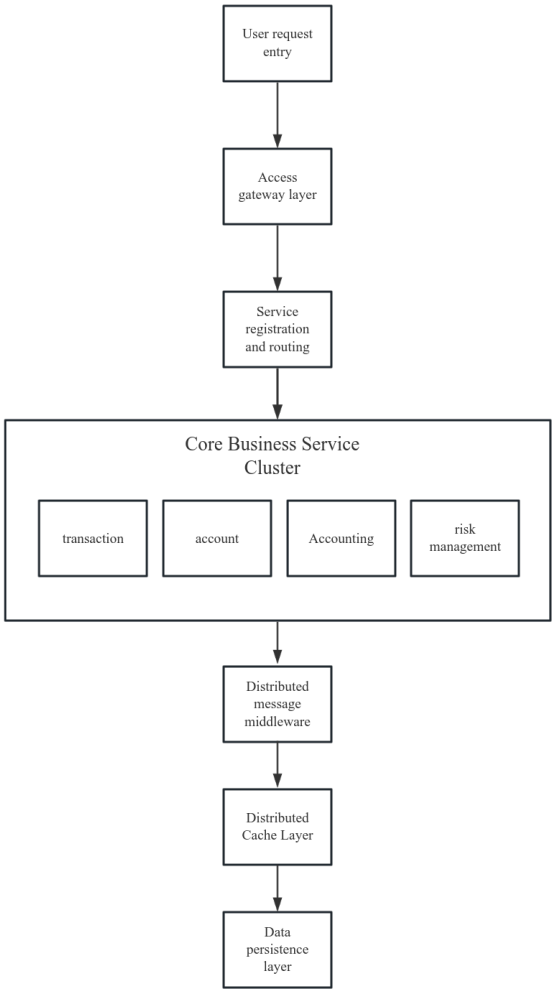

Figure 1. Typical distributed architecture flowchart of a high-concurrency payment system.

2.2. Consistency and Performance Conflict in Data Processing

In high-concurrency payment systems, there is a natural contradiction between system efficiency and data consistency [2]. On the one hand, payment requires a strong transactional nature, which means ensuring that data such as order status, account balance, and ledger records remain accurate during the transaction process to avoid data errors; On the other hand, to meet the high concurrency requirements, it is necessary to provide

higher throughput and lower latency response. However, as the system transitions from centralized to distributed design, this contradiction becomes increasingly prominent.

In centralized mode, consistency can be ensured through local transactions, with short execution paths and minimal performance damage. However, when dealing with multiple databases in a distributed environment, cross-node work is required to ensure data consistency, which can lead to a series of problems such as network latency and link instability. Therefore, different data consistency control schemes were adopted in the actual development process, such as 2PC, TCC, SAGA, message trading, etc. These mechanisms can achieve data consistency in different scenarios, but also introduce new trade-offs among system performance, availability, and development complexity.

To clearly demonstrate the technical characteristics and applicable boundaries of different consistency mechanisms, Table 1 compares and summarizes commonly used schemes:

Table 1. Comparison of Characteristics of Common Consistency Control Mechanisms.

Consistency mechanism	Conformance Levels	performance impact	Applicable scenarios	Overview of advantages and disadvantages
Local transaction (single node)	Strong consistency	High performance	Single system, small-scale transactions	Simple implementation, poor scalability
Two-stage submission (2PC)	Strong consistency	Low performance	Core accounting and balance processing	Reliable data, severe blockage, high failure rate
TCC model	Controllable consistency	Medium performance	Compensatable business, pre-authorized transactions	High idempotency requirements and strong business invasiveness
SAGA model	eventual consistency	High performance	Long transaction chain, order processing	Support concurrency and compensate for complex logic
Message transaction	eventual consistency	High performance	Asynchronous notifications, non-core writes	Consistent latency, dependent on message reliability

In high-concurrency environments, it is impossible to guarantee that the system will always run in a stable state. If secondary links can accept temporary inconsistencies, synchronization separation and final consistency should be the main approach to reduce the interference of consistency control on system performance.

3. Design Strategies for Data Consistency and Performance Scalability

3.1. Construction Plan for Hierarchical Consistency Guarantee Mechanism

To achieve a collaborative balance between data consistency and system performance in high concurrency scenarios, a consistency cost function can be introduced to quantitatively analyze the consistency strategies of different modules:

$$C_{total} = \sum_{i=1}^{n}(w_i \cdot D_i + \lambda_i \cdot L_i) \quad (1)$$

Among them, C_{total} represents the overall consistency control cost, D_i is the performance loss caused by the consistency strategy in the i module, L_i is the data synchronization delay of this module, and w_i, λ_i respectively represent the weight factors of performance sensitivity and delay tolerance of this module. By dynamically adjusting parameter combinations, a flexible configuration of consistency levels can be achieved under different business conditions.

At the beginning of developing this mechanism, each function was divided into three categories based on consistency requirements, such as account balance and payment confirmation modules [3]. A strong consistency strategy was used to ensure their correctness, and 2PC or Raft consensus protocols were used to ensure that multiple copies were written simultaneously, ensuring the consistency and immutability of transaction results; For modules with temporary inconsistency tolerance, a serializable consistency method is adopted, using MVCC and optimistic locks to achieve concurrent writing, reducing the blocking risk caused by synchronization while ensuring the correctness of business processes; The auxiliary flow functions such as logging and user behavior analysis are configured as the final consistency strategy, using asynchronous message queues to transmit data. By combining delayed write-backs, batch writes, scheduled scans, and other methods, the system's throughput is improved while ensuring the data availability within the consistency circle [4].

To enhance the reliability of institutional operation, pre-registration and idempotent identification control have been introduced to eliminate duplicate entries and chaotic states. Each action will be tracked with a unique ID to reduce redundant behavior and enable a general repair process. For unfinished work, it will be automatically retried, rolled back to the initial state, or manually intervened to ensure the recoverability and integrity of the entire process.

The above mechanism can improve the system's support for business consistency processing under overload conditions, enabling the system to have transaction reliability and elastic scalability balance, effectively supporting the elastic scheduling and high availability of the payment system.

3.2. Data Distribution Design for Concurrent Performance Improvement

The quality of the data distribution strategy determines the maximum concurrent quantity in concurrent payment systems. Therefore, based on the probability of concurrent conflicts and the nature of access volume, data should be horizontally segmented and optimized to construct dynamic paths and local load awareness. To quantify concurrency performance bottlenecks, a concurrency conflict overhead model is introduced:

$$T_{total} = \sum_{j=1}^{m} \left(\rho_j \cdot K_j + \theta_j \cdot R_j \right) \quad (2)$$

Among them, T_{total} represents the total delay of the system during concurrent processing, K_j represents the lock waiting time caused by conflicts in the j data node, and R_j is the read and write response time of that node; ρ_j and θ_j are the weight coefficients of conflict density and access frequency, respectively. By analyzing the model, high conflict and high-frequency access data segments can be identified as key optimization objectives in distribution design.

This system performs horizontal slicing processing through the business aspect, allocating transaction data related to different accounts, stores, and payment environments to specific servers using hash fields to reduce concurrent collisions [5]. For popular datasets, the method of hot-cold separation and hot path pre-positioning is adopted to ensure that frequent access paths have local hit capability and avoid burden on the main database.

In order to achieve dynamic load balancing, the scheduling module will dynamically adjust the replica routing based on the read and write rates and resource consumption of each partition node, achieving dynamic load balancing. Automatically back up the financial reports of high-traffic merchants to multiple busy servers during holidays or flash sales, and enable read-only path allocation to share the workload of writing and reading. The use of regionally distributed unique identifiers and a logical timestamp system avoids sequence number competition or chaotic write order caused by concurrent writes.

Build a two-level caching architecture: namely, the high-speed access hotspot data caching part and maintain data consistency through asynchronous updates; Simultaneously introducing write protection and update merge technology to reduce the pressure

on the main database. The handling of cross-node transactions adopts a method of submitting locally and confirming remotely to achieve data consistency, which can ensure that high concurrency does not interfere with the operation of the main process.

The overall architecture of the platform achieves data diversion, reduces hotspots and load balancing during high concurrency system access, improves the real-time processing capability and availability of the system, and provides data guarantee for stable operation during business peak periods.

3.3. Collaborative Optimization Path for Consistency and Scalability

To achieve a dynamic balance between consistency and scalability in high concurrency scenarios, the following performance trade-off model can be introduced:

$$U = \alpha \cdot S - \beta \cdot E \tag{3}$$

Among them, U represents the comprehensive performance score of the system, S is the service expansion capability of the system per unit time (such as throughput and node concurrency processing capability), E is the consistency maintenance cost (such as synchronization delay and write collision rate), $\alpha$and β are the scalability weight and consistency sensitivity coefficient, respectively. This model provides a quantitative evaluation framework for guiding the dynamic adjustment of consistency levels and system distribution strategies.

At the service design level, a strategy of module decoupling and data partitioning is adopted, and a state snapshot check is applied to ensure that all write actions are recognized by multiple backup instances. On non-transaction core routes, a combination of serial consistency management, MVCC technology, and local compensation algorithms is used to reduce the overall system transaction blocking rate. At the edge of the route, such as behavior recording, web browsing data, and analysis reports, the final consistency processing strategy is selected, and data is supplemented through asynchronous message delivery and scheduled tasks.

In interface design, based on the specific error handling logic implemented by the task, it is possible to choose which unified channel to use. Binding strongly consistent nodes to financial interfaces for atomic writing and transaction locking; The state class interface allows for synchronous backup or temporary cache reading, with timestamp and version information added for comparison and update; The information interface adopts fast screenshots of replicas and delayed library integration to provide real-time services to users, reducing the burden on the main library.

To achieve adaptive control during runtime, the scheduler dynamically tracks the performance of the system based on the above equation, and actively initiates a downgrade operation when efficiency decreases: turning some secondary connections into asynchronous write mode; making read operations on the server read-only. There are separation and conflict avoidance operations in the transaction flow. We have also built a fast rollback and log patching system to ensure the recoverability and verifiability of data status below the degraded consistency level.

This parallel path provides the end components with greater dynamic adjustment flexibility while ensuring the steady state of the main link, providing the system with an effective structural support for dynamic response to large shocks and rapid business adaptation and expansion.

4. Typical Case Analysis of 3 High-Concurrency Payment Systems

4.1. Asynchronous Consistency Transformation Case of a Certain Aggregate Payment Platform

The aggregate payment platform discussed in this case has a daily trading demand of over 100 million times, with a peak TPS of 96000 and an average response time of over 270ms. Additionally, the write channel blocking rate continues to rise, as account verification, point changes, SMS push, and other tasks are embedded in the business process,

seriously affecting the throughput of the main process and system availability. The comparison of key indicators before and after asynchronous consistency transformation is shown in Table 2.

Table 2. Comparison of key indicators before and after asynchronous consistency transformation.

Indicator items	Before refactoring	After refactoring
Average response time	270ms	112ms
TPS peak value	96 thousand	14.5thousand
Write blocking rate	High	↓82%
SLA availability	98.7%	99.999%

In the architecture reconstruction, the platform breaks down the processed payment-related content into a main chain and asynchronous event streams. The main chain handles deduction and financial storage, while the remaining content is sent through the Kafka cluster and asynchronously consumed by subsystems such as points, promotions, and settlements. By using an idempotent verification method, consumption state tracking, implementing a failure recovery retry strategy using Redis, and allowing message processing timeout within 3 seconds, the compensation success rate is 99.999%.

After the transformation, the average response time of the main chain decreased to 112ms, the write blocking rate decreased by 82%, the TPS peak was 145000 times, and the SLA increased to 99.999%. Some of the main performance indicators of the system have been significantly improved, indicating that the asynchronous adaptation of consistency to complex transaction processes has practical significance.

4.2. Concurrent Performance Optimization Case of a Certain E-commerce Flash Sale System

The e-commerce website's "Double 11" limited-time flash sale event has a high level of concurrent pressure, with a peak TPS of 21.7 thousand and a conflict rate of over 11% for inventory updates. The average time taken is over 300ms, and the SLA availability is only about 98.5%. The main issue is the performance bottleneck caused by lock contention in the centralized inventory deduction mode, which leads to high system concurrency. The comparison of key indicators before and after the transformation of the flash sale system is shown in Table 3.

Table 3. Comparison of key indicators before and after the transformation of the flash sale system.

Indicator items	Before renovation	After transformation
Peak TPS	21.7thousand	20thousand
Write conflict rate	>11%	<0.01%
Average response time	>300ms	92ms
SLA availability	About 98.5%	99.999%

To improve solving efficiency, the platform adopts an optimized architecture of "front-end token + local withholding + asynchronous confirmation". Before the activity is initiated, the inventory is cached in Redis, and each user's request is subjected to restricted traffic filtering before implementing the atomic subtraction operation locally and storing it in the Kafka queue. The backend consumers are asynchronously stored in the database, and optimistic lock anti-collision is introduced to automatically reverse the inventory value cached in Redis in case of conflicts.

After the system reconstruction, the peak TPS remained stable at 20 thousand, the inventory write conflict rate was reduced to less than 0.01%, the average response time for critical processes was significantly shortened to 92ms, and the SLA requirement for availability reached 99.999%. The system has significantly improved in parallel processing capability, write success probability, and service availability.

4.3. Data Expansion Architecture Reconstruction Case of a Financial Payment Channel

The financial service platform has access to over 90 banking service channels, and the original design has exposed significant bottlenecks in high-concurrency situations. The peak TPS of a single node is only 9000, the database storage latency is 210ms, and some transactions take more than 900s to process. The overall SLA availability is around 98.9%, making it difficult to adapt to the rapidly developing and exponentially growing demand for massive inputs and clearing. The comparison of key indicators before and after the reconstruction of the payment channel system is shown in Table 4.

Table 4. Comparison of key indicators before and after reconstruction of the payment channel system.

Indicator items	Before refactoring	After refactoring
Single-node TPS	0.9thousand	2.3thousand
Storage delay	210ms	82ms
Slowest delay in liquidation	>900s	180s
SLA availability	About 98.9%	99.999%

To enhance the functionality of the system and expand the entire system, the architecture of "channel segmentation + asynchronous access + database decoupling" is adopted to redesign the entire system. After separating transaction information according to different channel numbers, Kafka is used for asynchronous storage, Sharding JDBC is used for vertical database scaling, and balanced and independent channels are achieved during high concurrency with the support of several available web portals.

After optimization, the TPS of a single node has been increased to 2.3thousand, the data access time has become 82ms, the worst-case waiting time for settlement has become 180s, and the SLA guarantee rate has reached 99.999%. The key indicators of the system have been improved to varying degrees, enhancing the stability and scalability of the transmission channels in the financial industry.

5. Conclusion

This article focuses on the challenges of data consistency and performance scalability in high-concurrency payment systems. A solution covering hierarchical consistency management, data allocation optimization, and collaborative adjustment paths is proposed, and a quantitative guidance for strategy selection and architecture updates based on consistency and parallel efficiency cost models and parallel utility functions is established. In practical scenarios such as aggregated payments, flash payments, and financial channels, this method improves parallel performance while ensuring availability. The results indicate that in complex environments, the coordinated coexistence of consistency and scalability can be achieved through a module decoupling strategy, providing reference and theoretical support for the continuous evolution of highly available payment systems.

References

1. R. Ly, and B. Ly, "Digital payment systems in an emerging economy," *Computers in Human Behavior Reports*, vol. 16, p. 100517, 2024, doi: 10.1016/j.chbr.2024.100517.
2. V. Kesavan, and K. S. Srinivasan, "Present state and future directions of mobile payments system: a historical and bibliographic examination," *International Journal of Process Management and Benchmarking*, vol. 18, no. 3, pp. 329-351, 2024, doi: 10.1504/ijpmb.2024.142145.
3. A. K. Chalise, "Engineering and Advancement of Digitization through Unified Payment Interface," *Journal of Engineering Research and Reports*, vol. 24, no. 10, pp. 23-39, 2023, doi: 10.9734/jerr/2023/v24i10845.
4. M. Cipriani, L. S. Goldberg, and G. La Spada, "Financial sanctions, SWIFT, and the architecture of the international payment system," *Journal of Economic Perspectives*, vol. 37, no. 1, pp. 31-52, 2023, doi: 10.1257/jep.37.1.31.
5. B. Ramtiyal, D. Verma, and A. P. S. Rathore, "Interpretive structural modelling of factors affecting perceived risk in adoption of mobile payment system: a vendor's perspective," *International Journal of Intelligent Enterprise*, vol. 10, no. 3, pp. 264-279, 2023, doi: 10.1504/ijie.2023.131975.