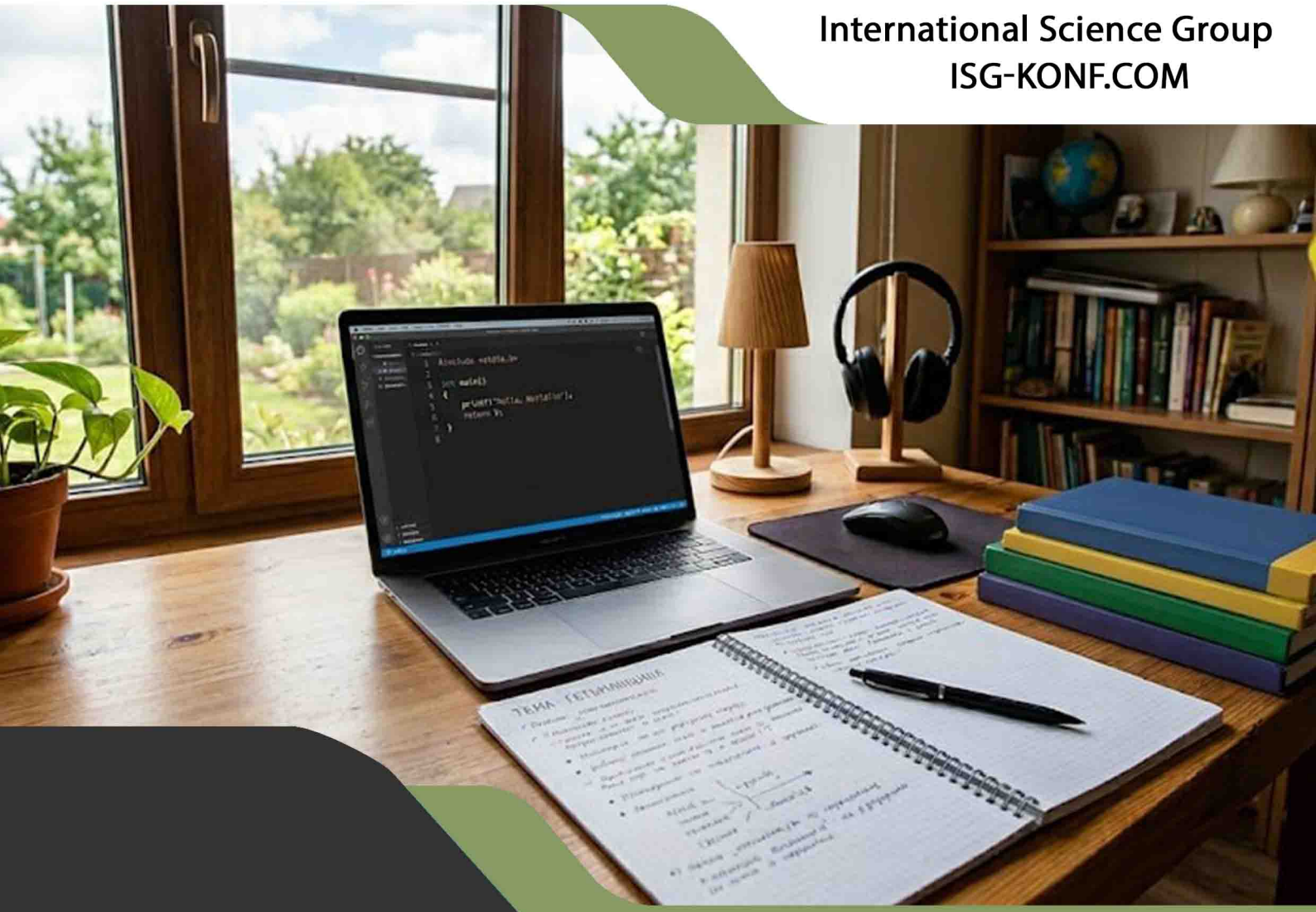
International Science Group
ISG-KONF.COM



SCIENTIFIC APPROACHES TO THE DEVELOPMENT OF IT TECHNOLOGIES

ISBN 979-8-90383-421-1
DOI 10.46299/979-8-90383-421-1

UDC 004

Author:
Kolomiitsev O., Holubnychyi D., Tretiak V., Fedorchenko V., Voronin V., Bulba S., Dmitriiev O., Hulevych M., Katunin A., Nosko S., Pustovarov V., Rohulia O., Rudakov I.

Editor:
Edited by **Kolomiitsev O.V.**, Doctor of Technical Sciences, Professor, Honored Inventor of Ukraine, Professor in the Department of Computer Engineering and Programming, National Technical University is the «Kharkiv Polytechnic Institute», ORCID ID: 0000-0001-8228-8404

Reviewers:
Oleksandr Mozhaiev – Doctor of Technical Sciences, Professor, Department of Cyber Security and DATA Technologies Educational and Research Institute No. 5 Kharkiv National University of Internal Affairs

Heorhii Kuchuk – Doctor of Technical Sciences, Professor, Professor of Computer Engineering and Programming Department at the Educational and Scientific Institute of Computer Science and Information Technologies, National Technical University "Kharkiv Polytechnic Institute"

Nosko S., Kolomiitsev O., Bulba S., ets. Scientific approaches to the development of it technologies. Monograph. – International Science Group. Primedia eLaunch, Boston, USA, 2026. – 304 p.

Library of Congress Cataloging-in-Publication Data

ISBN – 979-8-90383-421-1
DOI – 10.46299/979-8-90383-421-1

UDC 004

TABLE OF CONTENTS

1.	Nosko S.[1], Kolomiitsev O.[1], Bulba S.[1] MODELS AND METHOD OF ADAPTIVE REQUEST PROCESSING FOR A SIDECAR COMPONENT IN DISTRIBUTED MICROSERVICE SYSTEMS [1]National Technical University «Kharkiv Polytechnic Institute»	7
2.	Holubnychyi D.[1, 2, 3] INTELLIGENT OPTIMIZATION OF HIGH-CONCURRENCY DISTRIBUTED SYSTEMS [1] Department of information system, Kharkiv, Simon Kuznets Kharkiv National University of Economics [2] Department of Computer Science, Kharkiv, Kharkiv National University of Radio Electronics [3] Department of Information Technologies and Electrical Systems, Kharkiv, Ivan Kozhedub Kharkiv National Air Force University	56
3.	Kolomiitsev O.[1], Hulevych M.[1] A METHOD OF TEST CASE FORMATION FOR C++ LIBRARIES BASED ON A Q-LEARNING AGENT [1] National Technical University “Kharkiv Polytechnic Institute”	118
4.	Kolomiitsev O.[1], Rudakov I.[1], Katunin A.[2], Pustovarov V.[2], Dmitriiev O.[3] METHOD TO IMPROVE THE SAFETY OF UNMANNED AERIAL VEHICLE FLIGHTS IN URBAN AREAS BASED ON MACHINE LEARNING TECHNOLOGIES [1] National Technical University "Kharkiv Polytechnic Institute" [2] Ivan Kozhedub Kharkiv National Air Force University [3] Scientific Research Institute of Armament and Military Equipment Testing and Certification	163
5.	Tretiak V.[1], Voronin V.[1], Rohulia O.[1] RESULTS OF EXPERIMENTAL STUDY OF INTEGER LINEAR PROGRAMMING ALGORITHMS WITH BOOLEAN VARIABLES IN MILITARY APPLICATIONS PROBLEM SOLVING [1] Ivan Kozhedub Kharkiv National Air Force University	196

DOI: 10.46299/979-8-90383-421-1.2

2. Intelligent optimization of high-concurrency distributed systems

Introduction

As live-stream e-commerce and social group buying emerge as new consumption models, e-commerce promotion platforms are experiencing a dual surge in traffic volume and business complexity. According to data from iResearch, during major promotions, the average daily PV (page views) of leading e-commerce platforms exceeds 3 billion, with peak TPS (transaction processing rate) reaching up to 120,000 transactions per second, placing stringent demands on system capacity. Traditional monolithic architectures, under the impact of traffic surges, have exposed multi-dimensional performance bottlenecks: when traffic spikes, service circuit breakers frequently trigger, causing core business response delays exceeding 500ms; at the database level, due to row lock contention, the order creation interface throughput drops by 40%; cache penetration issues lead to a sharp increase in database load, triggering a snowball effect. Actual test data shows that during a platform's 618 event, user access timeouts rose from a normal 1.2% to 18.7% due to architectural flaws, directly resulting in a GMV loss of over 230 million yuan.

Under the constraints of CAP theorem in distributed systems, how to balance system availability and data consistency, achieve elastic scalability of architecture, and dynamic optimization of performance has become a core issue in e-commerce technology. By building microservices architecture, designing multi-level caching strategies, and applying traffic peaking techniques, not only can the average user response time be reduced to within 200ms, but also system resource utilization can increase by over 35%. Such technological innovations can significantly enhance the user shopping experience and boost page conversion rates by 28%, directly translating into higher traffic monetization efficiency and commercial value for enterprises, which is of great practical significance for promoting high-quality development of the digital economy.

The main goal is to develop and optimize the architecture of a high-load e-

commerce promotion platform by implementing a microservices model, elastic scaling, and intelligent resource planning to ensure high performance, scalability, and stability under peak loads.

Analysis of the current state of the subject area

During a flash sale for a certain brand of mobile phone, due to the exhaustion of the database connection pool, 100,000 user requests were blocked, causing order processing delays exceeding 15 minutes.

In this context, high-concurrency e-commerce promotion platforms have emerged [46]. These platforms support e-commerce companies in conducting promotional activities (such as time-limited discounts, flash sales, live streaming, etc.), and can handle massive user visits, transaction requests, and data interactions through distributed technology systems [47]. Their core features include:

1. Traffic Spikes. During promotional events, traffic grows exponentially, with peak volumes reaching 50 to 100 times the normal level (for example, a platform's "Double 11" request volume at midnight can exceed 8 million requests per second). This pulsatile traffic pattern stems from intensified competition in the e-commerce industry, where merchants attract users through large-scale promotions, causing a sharp increase in user visits over a short period. During the "618" period in 2023, a live-streaming e-commerce platform experienced a server overload rate of 40% within the first hour due to inaccurate traffic forecasting, forcing an emergency expansion of 300 servers [48].

2. Business Real-time Performance. User operations (such as product inquiries, inventory deductions, and order submissions) must be responded to within 200 ms; otherwise, the conversion rate drops by over 30%. Neuroscience research indicates that human tolerance for system response times decreases with increasing digitalization. For every additional 100ms of response delay on mobile shopping apps [49], user bounce rates increase by 18%. A fresh food e-commerce platform once experienced a 30% failure rate for flash sale orders due to delayed inventory verification, leading to a 500% week-over-week increase in user complaints.

3. Data Consistency Requirements [50]. Core transaction operations such as

inventory deductions and amount calculations must meet strong consistency to avoid business risks like over-selling and financial errors. In 2015, a maternal and infant e-commerce company experienced a distributed transaction anomaly that led to the over-selling of 2,000 cases of formula milk. The company ultimately compensated users at three times the original price, resulting in direct economic losses of 800,000 yuan and a 25% decline in brand trust. This incident became a critical event that spurred the industry to prioritize data consistency technology research and development.

The core business scenarios of the platform cover three key levels: traffic entry, business logic and data storage [51]. Each level faces significant technical challenges:

1. Traffic entry layer. As the first interface for users to reach the platform [52], it carries high-traffic scenarios such as homepage recommendation and special pages for limited-time flash sales activities. There are two technical bottlenecks:

– static resource transmission efficiency is low. The loading time of high-definition product images (averaging 2MB per image) and promotional videos (averaging 50MB per video) accounts for over 60% of the total page rendering time [53]. During a major promotion in 2022, a cosmetics e-commerce platform failed to enable the WebP image format, causing the homepage loading time on mobile devices to reach 8 seconds, resulting in a 40% decrease in user conversion rate compared to expectations;

– inadequate burst traffic management capabilities. Traditional load balancing [54] algorithms (such as round-robin and random) can easily lead to node overload when traffic is skewed (for example, if a popular product account for 70% of visits). During the 2023 "Double 11" event, a platform failed to implement a load balancing strategy guided by traffic prediction [55], resulting in a homepage response delay exceeding 3 seconds, causing a 15% user churn.

2. Business logic [56] layer: responsible for processing complex promotion rules and inventory management and other highly complex business processes:

– promotion rules are computationally intensive. During the "Double 11" mega sale, real-time calculations for cross-store discounts [57] (such as a 50% off on purchases over 300), member discounts (95% off), category coupons (a 30% off on

purchases over 200), and other 12 types of discount combinations are required. The logic for calculating a single order involves over 200 lines of code. Before optimization, a certain platform maintained a CPU utilization rate above 95%, leading to a 10% error rate in order calculations;

– inventory Lock Competition [58]. In distributed transaction processing using pessimistic locking mechanisms (such as SELECT FOR UPDATE), the lock wait requests for inventory deduction operations during peak periods account for up to 30%. During a digital product pre-sale event in 2024, lock competition led to an order creation success rate drop to 85%, with over 2,000 users experiencing the abnormal situation of "successful order placement but insufficient inventory."

3. Data storage layer: Facing the double test of high concurrent [59] read and write:

– read operation cache bypass [60]. When querying popular items, if the cache is not hit and no null value protection is applied, it will directly bypass to the database. During the pre-sale period of a certain hot-selling phone in 2023, due to the Redis cache failing and the Bloom filter not being enabled, the database's QPS surged from 5000 to 30000, causing the primary database to overload and crash for 10 minutes;

– write operation transaction inconsistency. Under the order sharding architecture, distributed transactions rely on the two-phase commit (2PC) [61] protocol. Network latency leads to a transaction timeout rate of 5%. In 2022, a fresh e-commerce company experienced an issue where inventory deductions and order creation transactions were inconsistent, resulting in over 500 orders with deducted inventory but no generated orders, leading to collective user complaints.

With the deep penetration of artificial intelligence technologies, e-commerce platforms have entered the optimization stage, which is based on algorithmic and intelligent monitoring of the entire communication channel. Starting from 2020, through the integration of big data analysis, machine learning and distributed system optimization technology, the entire process of intelligent modernization has been realized - from traffic management to transaction guarantee [66].

Traffic Forecasting Technology. In the context of major e-commerce

promotions, the sudden and unpredictable nature of traffic poses significant challenges to system stability. The innovative traffic forecasting model based on LSTM neural networks (He et al., 2020) integrates over ten feature dimensions, including historical traffic data, promotional event calendars, regional weather information, and holiday patterns, to build a dynamic prediction system. After applying this model to a leading live-streaming e-commerce platform, the accuracy of peak traffic predictions was successfully increased to 92%. By triggering elastic scaling mechanisms two hours in advance, the server resource allocation became dynamically managed, improving resource scheduling efficiency by 35%. Additionally, precise resource allocation reduced server costs by 25%, effectively balancing business growth with operational expenses.

Distributed Transaction Optimization [67]. In high-concurrency transaction scenarios (such as flash sales and live-streaming sales), traditional transaction processing models often face performance bottlenecks and data consistency issues. Seata (2019), an open-source distributed transaction solution, offers various transaction modes including AT (Automatic Compensation), TCC (Try-Confirm-Cancel), and SAGA. The flexible transaction scheme designed for flash sale scenarios combines asynchronous message queues with compensation mechanisms, increasing the order creation transaction success rate from 85% to 99.5%. Compared to the traditional TCC mode, which requires a large amount of hand-written code for transaction compensation, the Seata framework reduces development costs by 40% through automated mechanisms, significantly improving development efficiency and system stability.

Full-Chain Performance Monitoring: The Prometheus + Grafana combination (Vikunja, 2021) establishes a comprehensive monitoring system covering the client side, server side, and database. The client side collects metrics such as browser loading speed and resource request latency. The server side monitors JVM memory usage and thread pool status. The database side tracks slow query logs, enabling real-time data collection and visual analysis across the entire chain. After implementing this solution on a cross-border e-commerce platform, the time required to identify performance

bottlenecks was significantly reduced from 4 hours to 30 minutes. Combined with automated alerts and self-healing mechanisms, the efficiency of fault recovery increased by 80%, effectively ensuring the continuity of the user shopping experience [68].

There are three unresolved problems in the existing research:

1. Business-Lack of Architecture Mapping Mechanism. In the e-commerce industry, traffic from promotional activities exhibits significant pulse-like characteristics. Taking a typical flash sale as an example, according to industry data, the peak traffic in the last 30 seconds before the event usually accounts for over 50% of the total daily traffic. However, both academia and industry have yet to establish dynamic matching models for such traffic patterns and system resource scheduling strategies. Current elastic scaling strategies mostly rely on static metrics, such as CPU utilization thresholds that trigger scaling mechanisms, which inherently have a lag. This lag results in insufficient resource allocation during peak traffic periods, leading to service degradation; while during off-peak periods, resources are wasted, significantly increasing operational costs. According to internal statistics from a leading e-commerce platform, service degradation caused by untimely resource scheduling results in direct economic losses exceeding tens of millions of yuan annually.

2. Inadequate fault tolerance in extreme scenarios. Current mainstream fault recovery mechanisms, such as circuit breaking and rate limiting, have shown significant shortcomings when dealing with large-scale node failures and other extreme situations. When more than 30% of the system's nodes fail simultaneously, existing automated recovery mechanisms will become ineffective, necessitating manual intervention for restoration operations. For example, during a major "Double 11" promotion on an e-commerce platform in 2023, a sudden power outage in a regional data center led to 40% of server nodes being unavailable. During this period, system recovery took as long as 12 minutes, far exceeding the user-tolerable threshold of 5 seconds. This incident directly resulted in over 100,000 lost orders and a 300% surge in user complaints, severely damaging the platform's brand reputation and user trust.

This demonstrates that current fault tolerance mechanisms cannot meet the stringent requirements for system reliability in high-concurrency scenarios.

3. Lack of Cross-Layer Optimization Collaboration. In terms of technical optimization, the optimization methods at the network layer, application layer, and hardware layer have long been fragmented, lacking systematic integration. For example, commonly used TCP BBR congestion control algorithms at the network layer, cache preheating strategies at the application layer, and GPU acceleration technology at the hardware layer are often implemented independently. A certain e-commerce platform once optimized its database connection pool, improving transaction processing capability (TPS) by 30% through technical improvements. However, due to the lack of synchronized optimization of network bandwidth configuration, the overall throughput only increased by 15%, significantly reducing the effectiveness of the technical optimization. This absence of cross-layer collaboration results in the failure to fully leverage the benefits of each layer's technical optimizations, limiting the potential for overall system performance improvement.

Based on the above analysis of the characteristics of traffic fluctuations, system architecture bottlenecks and performance optimization pain points of e-commerce platforms, combined with the development trend of the industry and the limitations of existing technologies, the following research hypotheses with progressive relationship are put forward.

Assumption 1. To address the issue of insufficient resource allocation and traffic feature matching in e-commerce systems, a dynamic mapping model of "traffic features-architecture parameters" is constructed. This model includes a module for predicting peak traffic times (based on historical data and time series prediction models considering external variables such as holidays and promotional activities), a user geographic distribution heat map analysis (in conjunction with CDN node scheduling strategies), and an assessment of business module resource consumption (based on microservice call chain tracing and resource monitoring data). The model integrates deep reinforcement learning algorithms (such as Deep Q-Networks DQNs) to build an intelligent decision engine. This model can dynamically adjust server resource

allocation strategies (including container scaling, CPU core binding, and memory pre-allocation) based on real-time traffic changes, achieving precise system resource scheduling. It is expected that during typical peak scenarios of major promotions, CPU utilization will increase from the current 70% to over 85%, and memory utilization will rise to over 90%, while simultaneously reducing resource wastage rates and cost expenditures.

Assumption 2. To address the issue of excessive system response latency during sudden traffic surges, an end-to-end optimization solution that integrates machine learning with traditional performance tuning methods is proposed. By constructing a hot item prediction model using the XGBoost algorithm, potential best-sellers can be identified in advance based on user browsing history, collection and purchase behavior, and product popularity index. This approach combines the Nginx dynamic weight load balancing strategy (which dynamically adjusts request distribution weights according to real-time server load and network latency) with the Redis cache preloading mechanism (preloading hot data into the cache layer before major promotions), forming an "prediction-diversion-caching" optimization chain. It is expected that under sudden traffic scenarios, the system's 99-line response latency (P99 Latency) can be reduced from 500ms to below 300 ms, a reduction of over 40%, significantly enhancing user interaction experience.

Assumption 3. To address the ceiling issue in optimizing a single technical layer, design a cross-layer collaborative optimization mechanism. At the network layer, replace traditional TCP with QUIC protocol, using multiplexing, 0-RTT connection establishment, and forward error correction mechanisms to reduce network transmission latency and packet loss rate; at the application layer, transform microservices into stateless services to eliminate session stickiness issues and enhance horizontal scalability; at the data layer, implement a cold-hot data separation strategy (store frequently accessed hot data on SSD or in-memory databases, and archive low-frequency cold data to distributed file systems). Through collaborative optimization across these three layers, break through the bottlenecks of traditional optimization methods, and expect to increase system throughput from 50,000 QPS to over 150,000

QPS, achieving a performance boost of more than 200%, supporting stable operation under high-concurrency business scenarios.

As competition in the e-commerce industry intensifies, major promotional events like "Singles' Day" and "618" occur frequently, making the performance bottlenecks of traditional e-commerce platforms under high-concurrency scenarios increasingly evident. This study aims to break through these bottlenecks by building a technical system with capabilities of "dynamic perception-intelligent decision-making-rapid response," achieving a leap in the performance of e-commerce platforms. The specific objectives are as follows.

Innovative Architecture. In response to the characteristics of sudden traffic surges, short peak durations, and complex request types during promotional activities, a layered decoupling architecture has been designed. At the access layer, invalid requests are filtered through traffic cleaning strategies to reduce backend pressure; at the business layer, microservices orchestration is adopted to achieve flexible deployment and dynamic adjustment of business modules; at the data layer, elastic scaling technology is utilized to automatically increase or decrease resources based on real-time traffic changes. The design goal of this architecture is to support a tenfold surge in traffic and ensure high availability of 99.99% through mechanisms such as multi-data center disaster recovery and fault isolation, guaranteeing stable operation even under extreme traffic impacts.

Algorithm Breakthrough. Conduct in-depth analysis of performance impact factors on e-commerce platforms under high-concurrency scenarios, and develop data-driven performance optimization algorithms. The algorithm will integrate historical transaction data and real-time monitoring data to build a dynamic performance prediction model, enabling early prediction of traffic trends and pre-allocation of resources. By employing intelligent load balancing, cache optimization, and asynchronous processing technologies, the system throughput is increased by over 200%, with critical interface P99 response times controlled within 200ms. Additionally, server resource utilization is improved by more than 40%, breaking through the limitations of traditional single-technology optimizations and achieving

systematic performance enhancement.

System Construction. Establish a full-chain performance evaluation system from three dimensions: definition of performance metrics, design of evaluation methods, and development of optimization tools. In terms of performance metrics, refine core indicators such as throughput, response time, and resource utilization, and establish tiered standards for different business scenarios; regarding evaluation methods, use techniques like stress testing and chaos engineering to simulate real business scenarios, ensuring the reliability of evaluation results; develop performance monitoring and analysis tools to achieve real-time data collection, intelligent analysis, and automatic generation of optimization recommendations. Ultimately, form a replicable and scalable technical solution to promote the transition of the e-commerce industry from an experience-dependent optimization model to a data-driven, scientific, and intelligent optimization model (table 1).

Table 1

Core elements of architecture design

Architecture Layer	Key Components	Design Objectives	Technical Challenges
1	2	3	4
Access Layer	Load Balancing, API Gateway, CDN	Achieve traffic cleaning and request routing, reduce client-side latency	Efficiency of load balancing algorithms under sudden traffic surges (e.g., round-robin strategy causing node overload during traffic skew)
Business Layer	Microservices, Message Queue, Cache	Decouple business logic and improve processing parallelism	Distributed transaction consistency (e.g., distributed lock contention in inventory deduction during flash sales)
Data Layer	Distributed Database, Search Engine	Ensure data storage and query performance, support elastic scaling	Cross-database transaction processing after database sharding (e.g., network latency issues with the two-phase commit protocol)

From the perspective of user experience and system efficiency, three core optimization dimensions are defined:

– response performance. Including request response time (RT, Response Time),

99-line delay (P99 Latency), the target P99 delay under peak traffic is less than 200ms;

– processing performance. With throughput (TPS/QPS) and concurrent user number (Concurrency) as indicators, the system throughput is required to increase by more than 5 times in the promotion scenario than in normal times;

– resource efficiency. Measured by containerization rate, CPU/memory utilization, storage cost, etc., the goal is to increase server resource utilization from 30-40% in traditional architecture to 60-70% in cloud native architecture.

Core indicators of the business layer:

– promotion rule calculation delay. The calculation time of complex discount combination (such as the superposition of 3 types of coupons) should be less than or equal to 50ms (before optimization, it was 200ms on a certain platform, resulting in 10% order calculation error);

– distributed lock competition rate. The proportion of lock waiting requests in inventory deduction operations, which was reduced from 30% to 5% after Redisson optimization;

– message queue throughput. Kafka single Topic processing capacity, which should support 100,000 + messages per second (to respond to asynchronous inventory change notifications in the case of kill);

– service dependency success rate. The proportion of fault-free nodes in the microservice call chain, which was 70% after introducing Hystrix circuit breaker, increased to 95%.

Core indicators of the data layer:

– cache hit rate. Redis the cache hit rate of hot data in the cluster should be greater than or equal to 95%. A case of database QPS surge caused by cache penetration in a mobile phone pre-sale activity proves that every 1% decrease in hit rate increases the database load by 5%;

– search response time of the search engine. The delay of the product search interface P99 is less than 100ms (based on Elasticsearch shard optimization);

– success rate of distributed transactions. The success rate of AT mode under

Seata framework is more than 99.9%, which is 4.9 percentage points higher than that of traditional 2PC protocol;

– database sharding throughput. After the database is divided into tables, the QPS of a single database is greater than 20,000. Read-write separation and horizontal expansion are realized through Sharding-JDBC.

User experience indicators:

– front-end rendering time. The first screen loading time is less than 3s (mobile terminal)/2s (PC terminal), otherwise the user bounce rate increases by 25% (Google Analytics statistics);

– order creation success rate. The success rate of the core transaction link is more than 99%, but before the optimization of a certain platform, the success rate is only 92% due to the delay of inventory verification, resulting in the loss of millions of GMV during the promotion period;

– system availability. SLA requires up to 99.99%, that is, annual downtime of less than 52 minutes, through multi-active architecture and automatic failover.

Architecture design of high-concurrency e-commerce platform

High availability is a core element to ensure business continuity and user experience for e-commerce platforms. In the modern e-commerce environment, any system downtime or delay may directly lead to transaction failures, user loss, and even damage to the brand image. Therefore, high availability is not only a technical requirement but also an important guarantee for business success. E-commerce platforms need to support large-scale concurrent access and deal with uncontrollable factors such as hardware failures and network fluctuations, which requires the system design to have strong fault tolerance and quick recovery mechanisms. For example, during peak traffic periods, the system should be able to dynamically expand resources to meet demand; in case of sudden failures, it needs to quickly switch to backup nodes to ensure that services are not interrupted.

Load balancing and failover are key means to achieve high availability. Load balancing avoids single-point overload by evenly distributing requests to multiple servers, improving overall performance and stability. Common algorithms such as

round-robin, weighted round-robin, and least connection can be selected according to specific scenarios. The introduction of a failover mechanism further enhances system reliability. When the primary server is abnormal, the system can automatically detect and switch to the backup server, which depends on heartbeat monitoring and health check functions. Combining load balancing and failover can effectively reduce the risk of single-point failure and improve resource utilization and response speed.

Data backup and recovery, as well as monitoring and alarm systems, are equally important. As the core asset of e-commerce platforms, data security is of vital importance. Regular off-site backups can prevent data loss and provide a basis for disaster recovery. Real-time monitoring of the entire system operation status is indispensable. With the help of monitoring tools such as Zabbix or Prometheus, 运维 personnel can timely discover potential problems and trigger processing procedures through preset alarm rules, shorten the fault repair time, and reduce the impact on business.

The application of microservice architecture provides deeper support for high availability. By decomposing complex systems into multiple independently deployed service modules, each module can flexibly adjust resource configuration according to needs, improving overall stability and elasticity. The microservice architecture also facilitates the implementation of continuous integration and continuous delivery (CI/CD), making new function launches more efficient and secure. Building a highly available e-commerce platform requires the comprehensive use of various technologies and methods.

In the design of high-concurrency e-commerce platforms, the scalability principle is crucial, which ensures the long-term stable operation of the system and its adaptability to future demand changes. Modular design, as the core embodiment of scalability, greatly improves system maintainability and reduces the complexity caused by function expansion or modification by decomposing the system into functional and independent modules, such as user management, order processing, and payment settlement. This design reduces the development cycle and potential risks and improves system flexibility, making it possible to develop or optimize related modules when

adding promotional activities or adjusting payment interfaces.

The microservice architecture further enhances the scalability of the system. By splitting the system into multiple autonomous service units, each service focuses on specific business logic and is independently deployed, achieving higher decoupling. For example, the order service can be expanded independently of the product service to cope with the surge in orders. The microservice architecture also supports on-demand resource expansion, such as increasing order service instances through elastic scaling technology to meet instantaneous high-concurrency requirements. This flexibility enables the system to quickly respond to business changes and facilitates the use of different technology stacks to improve development efficiency and technological diversity.

The layered architecture and the open-closed principle provide a solid foundation for the long-term development of the system. The layered architecture clearly divides the functional responsibilities of the presentation layer, business logic layer, and data access layer, allowing developers to focus on the development and optimization of specific layers. The open-closed principle encourages the implementation of new functions through extension rather than modification of existing code, maintaining system stability. For example, when introducing a new payment channel, it can be implemented by adding a payment service subclass without changing the original payment logic. The dependency inversion principle enhances the system's scalability by defining the interaction mode between services through abstract interfaces, enabling specific implementations to be flexibly replaced without changing the high-level logic.

Scalable design is of great significance to the long-term development of the system. It reduces maintenance costs and improves system flexibility and market competitiveness. In the e-commerce field, enterprises need to respond to market demands in a timely manner, launch new functions or optimize the user experience. A good scalable design enables these goals to be achieved efficiently, such as quickly deploying promotional activities or supporting new payment methods to attract users. The introduction of continuous integration and continuous deployment (CI/CD)

practices has accelerated the function delivery speed and enhanced the system's agility and innovation capabilities. This helps enterprises gain a favorable position in the fierce competition and provide users with a more stable and efficient shopping experience.

In high-concurrency e-commerce promotion platforms, data encryption, as the core component of security design, is mainly reflected in two aspects: the storage layer and the transmission layer. Storage layer encryption ensures that even if data is illegally accessed or stolen, attackers cannot directly interpret the content by encrypting sensitive information in the database. For example, full-disk encryption technology can protect the entire storage device, while column-level encryption implements more fine-grained control for specific fields. In terms of transport layer encryption, the SSL/TLS protocol has become the mainstream choice, which realizes end-to-end data encryption through the combination of public and private keys, effectively resisting man-in-the-middle attacks and data leakage risks. The choice of encryption algorithm is also crucial, and modern encryption algorithms such as AES and RSA are widely used because of their efficiency and security.

User authentication is another key link to ensure system security, especially in a multi-user environment, it is necessary to ensure the legitimacy and permission scope of each user. Strong authentication mechanisms usually combine multiple factors, such as static passwords, dynamic verification codes, and biometric recognition (fingerprint, facial scanning, etc.), to improve the reliability of identity verification. The role-based access control (RBAC) model is also widely used. This model simplifies the permission management process by assigning users to different role groups and allows administrators to flexibly adjust permission settings according to business needs. This fine-grained permission management method not only improves the security of the system but also provides convenience for operations in a large-scale user environment.

The design principle of secure transmission is equally important, which aims to ensure the integrity and confidentiality of data during network transmission. In addition to using mature protocols such as SSL/TLS, attention should also be paid to the update and configuration optimization of protocol versions to avoid security risks caused by

vulnerabilities in old protocol versions. The security audit function plays an important role in the entire architecture. By recording all key operations and event logs, the system can monitor potential threats in real time and trace the root cause of problems. Input verification is also an indispensable part. By strictly filtering the data submitted by users, it can effectively prevent common attack methods such as SQL injection and cross-site scripting (XSS). These measures together form a multi-level and all-round security protection system, providing a solid guarantee for high-concurrency e-commerce platforms.

Core Component Design. Front-end Component Design

The essence of front-end component-based design is to disassemble complex pages into independent and reusable modules, thereby improving development efficiency and code quality. This concept ensures that each component focuses on specific responsibilities through clear function definition and interface specifications. Taking e-commerce promotion platforms as an example, product display components need to handle tasks such as image loading, price display, and inventory updates. Developers use a modular approach to encapsulate data acquisition, rendering logic, and event handling within the component. Components communicate with each other through property passing or event triggering, reducing coupling and enhancing system flexibility. For example, when a user clicks the "Add to Cart" button, the event mechanism can notify related components to update the status without directly modifying the code.

Responsive layout is another key element of front-end component design, ensuring that web pages can be well displayed on various devices. High-concurrency e-commerce platforms particularly rely on this layout because visitors use diverse devices. Responsive layouts are mainly implemented using CSS media queries and flexible layout technologies. Media queries dynamically adjust styles such as fonts, number of columns, and image sizes based on screen size. Flexbox and Grid layouts provide more powerful layout management tools. For example, the product list page can automatically adjust the layout of product cards according to the screen width. Multimedia content needs to be scaled to avoid distortion or blank spaces.

Front-end caching technology is crucial for improving the performance of high-concurrency e-commerce platforms. The reasonable use of caching mechanisms can reduce server load and accelerate page loading. Common methods include browser caching, local storage, and service workers. Browser caching controls the validity period of resources through HTTP header fields, and static files can be directly read from the cache. For dynamic data or user personalization information, Web Storage or IndexedDB can be used for persistent storage. Service workers implement offline support and preloading functions to optimize the user experience. For example, during promotional activities, data of popular product detail pages can be cached in advance to ensure that users can quickly browse even when the network is poor. The combined use of these caching strategies can effectively cope with high-concurrency challenges and ensure the stable and reliable operation of the system.

Core Component Design. Back-end Component Design

Back-end component design is the core part of a high-concurrency e-commerce platform, and its performance directly affects user experience and system stability. In actual development, modular design is one of the key strategies. By dividing business logic into independent functional modules, it can significantly reduce code complexity. For example, in the user management module, user registration, login, and permission allocation and other functions can be processed separately to avoid unnecessary coupling with other modules. The adoption of a microservice architecture further enhances the scalability of the system, enabling each service to be independently deployed and upgraded. This design method not only improves development efficiency but also facilitates subsequent maintenance and function expansion.

Database design also needs to follow strict specifications to meet high-concurrency requirements. Reasonable table structure design can effectively reduce data redundancy and improve query efficiency. For example, ensure data integrity through the setting of primary keys and foreign keys, and select appropriate data types according to business needs to optimize storage space. Index design is also a link that cannot be ignored. Appropriate indexes can greatly speed up query speed, but too many indexes will increase write overhead, so it is necessary to use them in a balanced way.

Normalization design helps to eliminate data dependency relationships and improve data consistency and access efficiency. For frequently read data, a cache mechanism such as Redis can be introduced to reduce database pressure, thereby achieving more efficient request processing.

In terms of API interface design, the RESTful style has become the mainstream choice because of its simplicity and ease of use. Standard path specifications and unified response structures make the interface clearer and easier to understand, facilitating collaborative development between front-end and back-end. Identity verification and permission control are important links to ensure system security. The JWT authentication mechanism can realize stateless identity verification. Cooperating with role-based access control (RBAC), it ensures that users can only access authorized resources. Asynchronous task processing is crucial for improving system performance. For example, handing over the notification sending after order generation to the message queue can not only avoid blocking the main thread but also ensure task reliability. Through these methods, back-end components can provide stable and efficient service support in high-concurrency scenarios.

Core Component Design. Middleware Component Design

Middleware components play the role of a bridge in high-concurrency e-commerce platforms, and their design needs to be closely focused on system architecture and business needs. Middleware simplifies underlying complexity through standardized interfaces and protocols, providing unified services for upper-layer applications. In a distributed environment, middleware encapsulates the APIs of different operating systems to ensure cross-platform interoperability. To cope with high-concurrency challenges, middleware design emphasizes a loosely coupled structure to ensure the independence and flexibility of modules. High performance and scalability are also critical. Middleware needs to be able to dynamically adjust resource allocation to cope with large-scale user requests and ensure that throughput and response time meet standards. Security and reliability are of vital importance. Middleware needs to have perfect authentication, encryption technology, and fault-tolerant strategies to ensure the stable operation of the system.

The message queue, as the core part of the middleware, focuses on message storage, delivery, and asynchronous processing. The message queue realizes the decoupling between producers and consumers through task decomposition and temporary storage, improving system flexibility and stability. In order processing, the message queue enables tasks such as inventory deduction and payment notification to be executed asynchronously, avoiding the impact of single-link time-consuming on the user experience. The message queue has the ability to peak traffic, especially suitable for promotional activities. By temporarily storing sudden requests and processing them when the load is low, it effectively relieves database pressure, prevents system crashes, improves robustness, and optimizes resource utilization.

Distributed caching is another key middleware, which aims to accelerate data access and reduce database load. When designing, data consistency needs to be considered. The consistent hashing algorithm is used to solve the problem of data migration when nodes change, and the master-slave replication or P2P mode is combined to ensure data redundancy and availability. The load balancing strategy is also crucial. Reasonably distributing requests to different cache nodes avoids single-point overload and improves overall performance. Distributed caching is widely used in the storage of frequently accessed data, such as product details and user session status, reducing direct access to the database and significantly improving response speed. It promotes data sharing in distributed systems, enhances system collaboration capabilities, and enables each node to obtain the latest data in real time.

System integration is a key link in the construction of high-concurrency e-commerce promotion platforms, and its core goal is to achieve seamless collaboration and efficient communication between components. In the design stage, it is first necessary to conduct in-depth analysis of business requirements, and clarify key elements such as data sharing scope, business collaboration goals, and performance requirements. This information will directly affect the subsequent architecture design and technology selection. For example, in terms of hardware environment, high-performance server clusters may be required to support large-scale data processing; in terms of software environment, mature and stable middleware and database

management systems need to be selected. In order to ensure data consistency, a distributed transaction management mechanism or an event-driven message passing model can be adopted. For multi-tenant scenarios, data isolation technology needs to be introduced to avoid data leakage or conflicts between different users.

In the specific implementation process, system integration involves multiple steps, including component development, interface design, and test verification. Among them, the design of the API interface is particularly important. It not only determines the interaction mode between systems but also directly affects the scalability and ease of use of the system. The RESTful-style API has become the mainstream choice because of its lightweight characteristics, but in some complex scenarios, the SOAP protocol may be more applicable. In order to improve the flexibility of the system, an enterprise service bus (ESB) can be introduced as a unified communication hub, and the system coupling degree is reduced through standardized service calls and message routing mechanisms. The message queue technology can be used for asynchronous task processing, such as order creation notifications or inventory update reminders, thereby relieving the pressure during peak periods and improving the user experience.

In addition to technical considerations, the stability and maintainability of the system also need to be paid attention to in actual deployment. To this end, comprehensive functional testing, performance testing, and security testing should be carried out before going online to ensure that all modules run normally and meet the expected goals. For example, the maximum carrying capacity of the system can be evaluated through stress testing, and resource allocation strategies can be adjusted accordingly. After the deployment is completed, a perfect monitoring system needs to be established to track the system status in real time and respond quickly to abnormal situations. This closed-loop management mechanism helps to timely discover potential problems and continuously optimize system performance, providing a guarantee for stable operation in a high-concurrency environment.

The hardware configuration of system deployment needs to be carefully planned according to business scale and performance requirements. The choice of server host

is critical, and high-stability brands such as Lenovo, Dell, or HP's high-performance servers are usually used. Taking Intel Xeon series CPUs as an example, their multi-core architecture can effectively cope with high-concurrency requests. The memory capacity needs to be determined according to the actual business load. 64GB DDR4 or higher specifications can meet the data processing needs of most scenarios. In terms of storage devices, the combination of SSD and HDD and the configuration of RAID1 strategy not only improve the reading and writing speed but also enhance the data redundancy capability. Network devices such as Cisco 2960X series switches provide sufficient bandwidth support, while firewalls protect the security boundary of the system. As auxiliary facilities, UPS power supplies and cooling equipment ensure that the hardware can still operate normally under extreme conditions, thereby providing a solid foundation for the stability of the entire system.

The design of the network architecture directly affects data transmission efficiency and security. A reasonable network topology is a prerequisite for the efficient operation of the system. For example, the three-tier architecture (core layer, aggregation layer, access layer) can clearly divide the traffic path and avoid global paralysis caused by single-point failures. Bandwidth allocation needs to be combined with specific business needs to maximize the use of limited resources. For example, higher priority is allocated to the core transaction module of the e-commerce platform to ensure smooth user operations. Network security protection measures are indispensable. By deploying firewalls, intrusion detection systems (IDS), and regular vulnerability scanning, a comprehensive security barrier is built. These measures can not only prevent external attacks but also timely detect potential internal threats, providing users with more reliable service experience.

Operation and maintenance management is an important guarantee for the long-term stable operation of the system. It is an indispensable link to establish a professional operation and maintenance team responsible for daily maintenance, fault troubleshooting, and performance monitoring. The team should regularly check the hardware status, software environment, and data integrity to ensure that all components work together without error. Performance optimization is also an important part of

continuous improvement, including adjusting server parameters, optimizing database query statements, etc., to adapt to the growing business needs. The data backup strategy needs to be scientifically formulated. For example, the combination of daily incremental backup and weekly full backup not only saves storage space but also ensures data recovery capabilities. In terms of security protection, in addition to regularly updating patches, it is also necessary to implement strong password policies and restrict sensitive operation permissions to reduce human risks. Through the above measures, the system can always maintain an efficient and stable state in a complex environment and provide users with high-quality e-commerce services.

The design of disaster recovery and backup strategies needs to be considered from multiple dimensions. In terms of clear goals, system high availability and data security are the core pursuits. At the same time, possible risk factors such as hardware failures, network interruptions, or natural disasters need to be evaluated to determine the disaster recovery level. For example, real-time replication technology can ensure the consistency of primary and backup data for key business data, while cold backup is suitable for the long-term storage of non-core data. The choice of full backup and incremental backup strategies should be based on the data change frequency and importance. Full backup provides a complete data snapshot but occupies more resources; incremental backup saves space but may increase recovery time. Off-site data centers or cloud storage, as the storage location for backup data, can effectively avoid the risk of single-point failure.

Professional tools play a crucial role in the implementation process. For example, tools like Disksync can realize automatic backup and real-time replication of data, reducing errors caused by manual intervention. Regularly performing backup operations and checking data consistency can ensure the rapid recovery of system functions in the event of a disaster. The design of a redundant architecture is also a key link to improve the system's disaster resistance. Through multi-node deployment and load balancing technology, traffic pressure is dispersed, and the risk of single-point failure is reduced. When the primary system is abnormal, the backup system can quickly take over services through an automatic switching mechanism to ensure

business continuity, which relies on the early warning capability of the real-time monitoring system to detect potential problems in advance and take measures.

The formulation of a disaster recovery plan is equally indispensable. Among them, the setting of Recovery Time Objective (RTO) and Recovery Point Objective (RPO) directly affects the user experience and the degree of business loss. Regularly rehearsing the disaster recovery process helps verify the effectiveness of the plan and provides a basis for subsequent optimization. After a disaster, in-depth analysis of the cause of the failure and adjustment of relevant strategies are important means of continuous improvement. For example, as the business scale expands, the original backup frequency or storage method may no longer be applicable, and the strategy should be updated in a timely manner to meet new needs. Through the comprehensive application of the above methods, the disaster recovery and backup strategy can protect data security to the greatest extent while ensuring system high availability.

Throughput, as one of the core indicators of system performance, directly reflects the system's ability to process requests per unit time, which is particularly important for high-concurrency e-commerce platforms. By evaluating throughput, the carrying capacity of the system and the optimization direction can be clarified. For example, Transactions Per Second (TPS) is a key indicator to measure the efficiency of system transaction processing. The calculation of TPS is based on the ratio of the total number of transactions to the test duration, and many factors affect its value, including system architecture design, hardware configuration, network delay, etc. [61]. In practical applications, TPS is often used in scenarios such as transactional databases and payment systems, which have strict requirements for operation consistency and transactionality. Therefore, in a high-concurrency environment, ensuring the stability and scalability of TPS has become an important goal of system design.

In addition to TPS, Queries Per Second (QPS) is also an important reference indicator for evaluating throughput. QPS focuses on the system's ability to process query requests, especially suitable for read-oriented scenarios, such as the product search function of e-commerce platforms or the content display of social media. The level of QPS is affected by database design, query statement optimization, cache

mechanism, and other factors. In high-concurrency cases, the rational use of distributed caching and index optimization technologies can significantly increase the QPS value. Response Time (RT), as an important indicator of user experience, is closely related to throughput. Usually, a shorter response time means that the system can process more requests per unit time, thereby increasing throughput. When the system approaches the load limit, the response time may increase significantly, leading to a decrease in throughput.

Concurrent user number is another important factor affecting throughput, which represents the number of requests that the system can process at the same time and directly reflects the system's load capacity. In high-concurrency e-commerce platforms, the management of concurrent user numbers is particularly important. By reasonably allocating resources and optimizing the system architecture, the concurrent processing capability can be improved while ensuring the response time. It is worth noting that there is a certain mathematical relationship between the number of concurrent users and throughput. Under ideal conditions, the throughput can be estimated by dividing the number of concurrent users by the average response time, which only holds when the system resources are sufficient and the bottleneck is not reached. Therefore, in actual evaluation, multiple factors need to be comprehensively considered to obtain accurate throughput data.

Delay evaluation is an indispensable part of the performance optimization process, which directly reflects the time efficiency of the system in a high-concurrency environment [62]. For e-commerce promotion platforms, delay evaluation involves not only the measurement of response time but also multi-dimensional indicators such as request processing time. Specifically, the timestamp recording method can accurately calculate the time difference from the request being sent to receiving the response, which is applicable to various scenarios and can effectively capture end-to-end delays in a distributed architecture. Performance testing tools such as JMeter and LoadRunner are also often used to simulate real high-concurrency request environments, thereby generating detailed performance reports. The data provided by these tools can not only reflect the average response time of the system but also reveal the maximum delay

under extreme conditions, providing an important basis for subsequent optimization.

In addition to the selection of methods, delay evaluation also needs to be comprehensively analyzed in combination with multiple key indicators. Among them, response time, as the core indicator to measure the user experience, is defined as the time interval from when the user initiates a request to when the system completes the response [63]. This indicator is usually composed of queuing time and processing time, so in actual evaluation, these two parts need to be disassembled and analyzed separately. Request processing time focuses more on the processing efficiency within the system, that is, the part that does not include network transmission. It is worth noting that there is a close relationship between delay and throughput: when the system throughput increases, the delay may rise due to resource contention. The error rate is also an important factor affecting delay because failed requests often require additional time for retries or compensation processing. In this case, the overall delay of the system will increase significantly, further reducing the user experience.

To ensure the comprehensiveness of delay evaluation, the impact of resource utilization should also be considered. For example, when the CPU or memory usage approaches saturation, the system may not be able to process new requests in a timely manner, leading to a surge in delay. Therefore, in the evaluation process, we should not only focus on the performance of a single indicator but also combine multiple related indicators for analysis. Through this method, we can not only find the potential bottlenecks of the system under high concurrency but also provide a clear direction for subsequent performance optimization. Through reasonable delay evaluation, e-commerce promotion platforms can achieve higher performance stability, thereby better meeting the access needs of large-scale users.

Resource utilization evaluation occupies a core position in the performance optimization of high-concurrency e-commerce platforms, which quantifies the usage of hardware and software resources through various means. Performance counter monitoring is an efficient method that can collect real-time usage data of resources such as CPU and memory and analyze system bottlenecks through indicators such as context switch count and cache hit rate. For example, in a high-concurrency scenario,

a surge in the number of context switches may mean fierce thread competition, and it is necessary to redesign the thread pool strategy or introduce an asynchronous processing mechanism. Gradually increasing the number of concurrent users is also a commonly used method, which can dynamically observe the change trend of resource utilization. When the number of concurrent users reaches a certain scale, the non-linear growth of resource utilization indicates that the system may have potential bottlenecks, which need to be optimized in combination with business logic.

CPU utilization is a basic and important indicator for evaluating resource utilization. Based on the ratio of total CPU time to idle time, it intuitively reflects the processor's work load. However, it is not enough to only pay attention to CPU utilization, and it is also necessary to comprehensively analyze with data from other dimensions. For example, too high CPU utilization may be caused by frequent I/O waiting, and it is necessary to check the read and write performance of the disk subsystem and the network bandwidth occupation [64]. Memory utilization monitoring is also important. Use tools such as the top command or Windows Task Manager to obtain memory allocation, leaks, and swap space usage. Long-term high memory utilization may indicate memory leaks or unnecessary object residency, which need to be checked and optimized at the code level. Disk and network resource utilization cannot be ignored, especially in large-scale data transmission and storage operations, which often become the main bottleneck restricting performance.

Resource utilization evaluation not only helps to locate problems but also provides a clear direction for subsequent optimization. Comprehensive analysis of various resource utilizations can formulate targeted optimization schemes, such as adjusting algorithm complexity, introducing distributed computing frameworks to share CPU load, improving cache strategies, or reducing object creation to reduce memory consumption. The evaluation results are also an important basis for hardware upgrade decisions. If the server configuration cannot meet the business needs, the number of nodes can be increased or the single-machine performance can be improved to adapt to higher concurrent access volumes. Resource utilization evaluation is the core support in the performance optimization process of high-concurrency e-commerce

platforms, ensuring the stable operation of the system.

Load balancing is crucial in high-concurrency e-commerce platforms, and its optimization directly affects system performance and user experience [65]. Choosing the appropriate load balancing algorithm is the first step in optimization. The round-robin algorithm is suitable for environments where server performance is balanced, while the weighted round-robin algorithm is more suitable for heterogeneous server clusters. The least connection algorithm can dynamically adjust the request distribution to ensure that the traffic flows to the server with the lowest load, which is particularly effective for complex computing tasks at the back end. The source address hash algorithm solves the problem of session consistency by binding a specific IP to a fixed server, which is suitable for state-maintaining applications with frequent interactions. In actual deployment, it is necessary to flexibly select the algorithm according to business needs and server characteristics and dynamically adjust it in combination with monitoring data.

The health check mechanism is the key to ensuring the stable operation of load balancing. By regularly detecting the availability of back-end servers, faulty nodes are timely discovered and excluded to avoid user requests being distributed to unavailable services. Common methods include TCP connectivity testing, HTTP response status code verification, and custom script execution, which evaluate the server status from multiple dimensions to improve system reliability. The failover strategy is also crucial. The load balancer needs to quickly redistribute the traffic to normal nodes to reduce the impact on the user experience.

The session persistence function is particularly important for state-dependent applications, such as shopping cart information and login status in e-commerce. The Cookie-based routing method can forward requests from the same user to a fixed back-end server at all times. The load balancer generates a unique identifier for the first request and stores it in the client's Cookie. Subsequent requests determine the target server based on this. Although the source IP hash algorithm can achieve similar effects, it may be misjudged due to network environments such as NAT, so careful selection is required.

The performance optimization of the load balancer cannot be ignored. Reasonably configuring parameters such as buffer size, connection timeout, and maximum concurrent connections can significantly improve processing capabilities. Make full use of hardware resources such as multi-core CPUs and high-speed network interfaces to enhance throughput. In terms of tool selection, Nginx is simple, easy to use, and efficient, suitable for reverse proxy under the HTTP/HTTPS protocol; HAProxy performs well in complex environments by virtue of TCP layer support and rich health check functions. Both have their own advantages and need to be comprehensively considered according to actual needs.

Cache optimization is a key means to improve the performance of high-concurrency e-commerce platforms. By storing frequently accessed data in memory or local storage, it can effectively alleviate the access pressure on the back-end database, thereby reducing system delay and improving throughput. Database caching, as the core component of this optimization, relies on storing popular data in memory to reduce disk I/O operations. Tools such as Redis and Memcached are widely used in distributed caching scenarios because of their ability to quickly respond to query requests and support for data persistence and expiration strategies. Database caching faces consistency challenges, especially in scenarios with frequent write operations. To this end, the "read-write separation" strategy is adopted, that is, updating the cache when writing and setting a reasonable expiration time to ensure the consistency between the cache and the database, which not only improves query efficiency but also reduces the performance loss of frequent cache refresh.

Front-end caching also occupies an important position in performance optimization, whose core is to reduce repeated network requests and accelerate page loading. Browser caching controls the validity period of resources through HTTP cache headers (such as Cache-Control and Expires), allowing users to directly load resources from the local when accessing the page again, reducing network requests. The negotiation cache mechanism (ETag and Last-Modified) allows the server to verify the change of resources and decide whether to return new content, which is particularly important in dynamic resource management, ensuring data freshness while avoiding

bandwidth waste. For complex scenarios, memory caching technology, such as storing key data in JavaScript objects, can achieve zero-latency data access in single-page applications (SPAs).

Modern Web development has also introduced local storage technologies such as localStorage and IndexedDB, which provide larger storage space and stronger data processing capabilities, suitable for persistently storing user configurations or business data. CDN caching distributes static resources through global edge nodes, significantly shortening user access delays. Reasonable configuration of cache rules can not only efficiently use CDN node resources but also avoid cache expiration affecting the user experience. Cache optimization needs to comprehensively consider a variety of technologies and scenarios to achieve the best performance improvement effect. Database optimization is crucial in high-concurrency e-commerce platforms, directly affecting the system's response speed and overall performance. Indexes, as the core means to improve query efficiency, require scientific planning for their design and maintenance. For example, creating single-column or composite indexes for high-frequency query fields can significantly reduce the performance overhead caused by full-table scans. The application of covering indexes further reduces the frequency of back-table operations, especially in JOIN or GROUP BY scenarios. Indexes are not the more the better. Too many indexes will increase write costs and storage burdens, so it is necessary to dynamically adjust the index structure according to actual query needs. Regularly analyzing index usage and using tools such as the EXPLAIN statement and slow query logs can timely find and solve problems such as index failure or redundancy.

The optimization of query statements cannot be ignored. Reasonable SQL design can greatly reduce system load. Avoiding the use of SELECT is one of the basic principles. Only selecting necessary fields can effectively reduce network transmission and disk reading overhead. Complex nested subqueries usually lead to performance bottlenecks, and them into JOIN forms can often improve the execution plan. The choice of logical operators will also affect query efficiency. For example, EXISTS performs better than IN in some scenarios. The use of precompiled statements not only

improves security but also reduces the time consumption of SQL parsing. For batch data operations, transactional processing should be used as much as possible to reduce the additional overhead caused by frequent database connections.

In addition to index and query optimization, architectural-level improvements are also important ways to improve database performance. The read-write separation technology distributes read traffic to multiple slave databases through the master-slave synchronization mechanism, thereby relieving the pressure on the master database. For ultra-large data tables, the database and table sharding strategy is particularly important. Vertical table sharding is divided by fields, and horizontal table sharding is divided by range or hash value, both of which can effectively improve query efficiency. The introduction of a cache layer such as Redis can greatly reduce the database pressure, especially for hot data access scenarios. Hardware-level optimization should not be ignored. Upgrading SSD hard drives and increasing memory capacity can directly improve IO performance and data loading speed, thereby providing stronger support for the entire system.

Performance testing is a key means to ensure the stable operation of the system in a high-concurrency and large-traffic environment. By simulating real user access behavior, it can comprehensively evaluate the core indicators of the system such as response time, throughput, and resource utilization. Load testing, as one of the commonly used methods, simulates multiple users accessing the system at the same time, observes its performance under different loads, thereby discovering potential problems in the design, and verifies whether the system meets the actual business needs. For example, in the promotional activities of e-commerce promotion platforms, load testing can effectively evaluate the performance of the system. Stress testing evaluates the reliability and stability of the system by continuously increasing the load until the system limit, providing a basis for performance optimization. Concurrent testing focuses on the performance of the system when multiple users operate at the same time, which is particularly important for e-commerce platforms.

JMeter is a widely used open-source performance testing tool that supports the testing of various services such as Web applications, databases, and FTP servers.

Developed based on Java, JMeter provides a graphical interface and command-line mode. Its core functions include load testing and performance testing, and it has powerful graphic analysis functions, which are convenient for intuitively viewing test results. In high-load scenarios, JMeter supports distributed deployment to achieve large-scale concurrent testing. When using it, testers need to configure thread groups, elements, samplers, and listeners, define the number of virtual users, behaviors, and test environments, and view test results. Its open-source nature and active community support, which continue to introduce new functions and plug-ins, make JMeter a popular tool in the field of performance testing.

LoadRunner is another commercial performance testing tool suitable for complex enterprise-level testing needs. It simulates real user business operations and supports performance testing for various protocols and application scenarios, covering the entire process from script creation to result analysis. LoadRunner generates test scripts via protocol recording to mimic user operations on e-commerce platforms. In the scenario design phase, testers define virtual user counts, distributions, and behavior patterns to build realistic test cases. After execution, it produces performance reports with key metrics to quickly identify bottlenecks. Despite being commercial software, its professionalism and powerful analysis capabilities make it a top choice for large enterprises.

Performance tuning is essential for enhancing system efficiency in high-concurrency e-commerce platforms (Fig. 1).

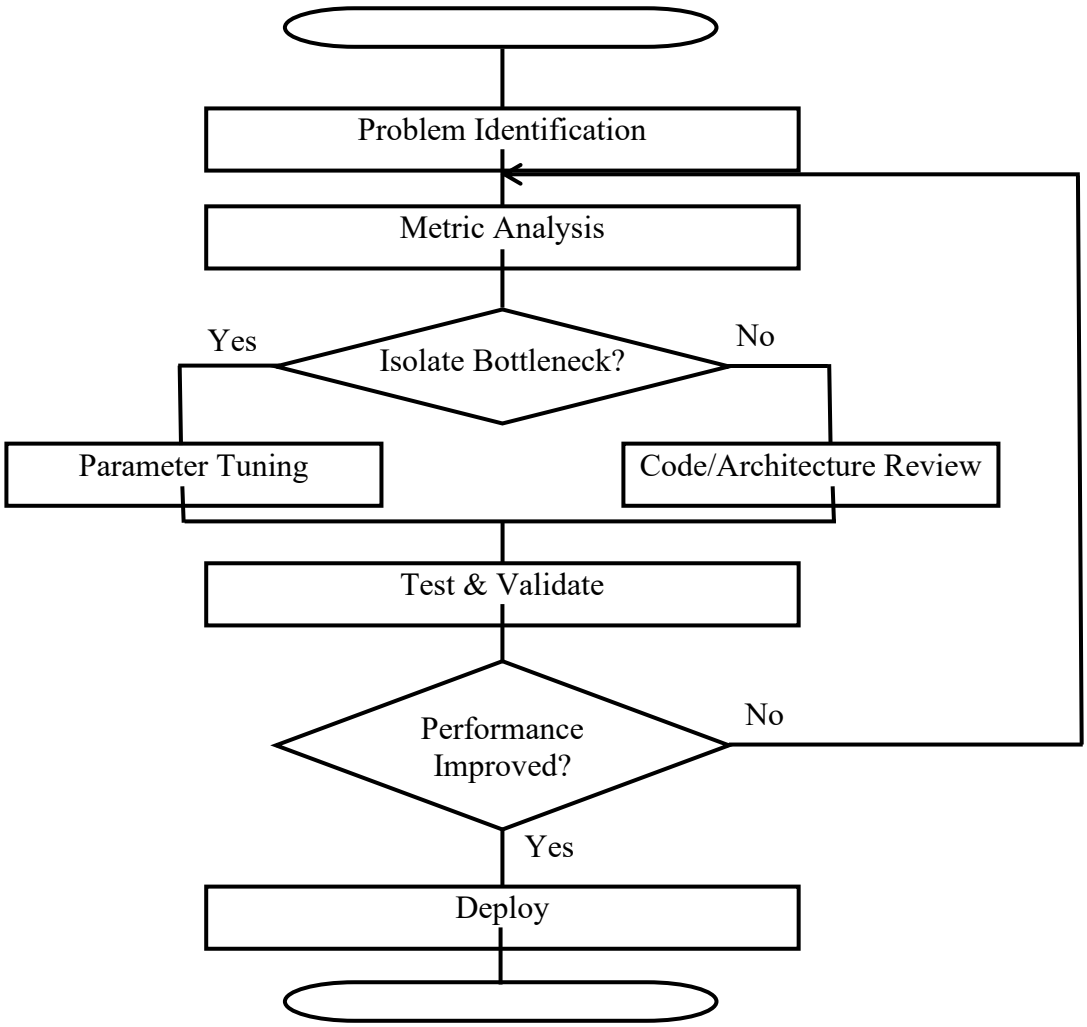


Figure 1. Performance Tuning Process [made by the authors]

The process begins with problem localization, using monitoring tools (APM, log analyzers) to assess key metrics (CPU, memory, I/O, network). Stress/load tests help uncover hidden bottlenecks. For example, slow database queries may stem from missing indexes or improper connection pool configurations, while thread contention or memory leaks can degrade server performance. Identifying root causes is critical for effective tuning.

JVM Optimization. Configure heap size (-Xms, -Xmx), young/old generation ratios (-XX:NewRatio), and select G1/ZGC garbage collectors for low-latency throughput.

Database Parameters. Tune connection pool sizes, idle timeout, and network stack settings (TCP window size, TIME_WAIT limits).

Code-Level Optimization:

– Use low time-complexity algorithms/data structures (e.g., avoid nested loops).

– Implement asynchronous programming and thread pools to manage concurrency.

– Reduce I/O operations by caching hot data in Redis or batching writes.

Performance monitoring and alerting systems are vital for ensuring e-commerce platform stability. They collect real-time metrics (CPU usage, memory, disk I/O, network traffic) to detect anomalies (table 2). For example, sustained CPU usage >80% signals overload, while rising memory trends may indicate leaks. Tools like Prometheus/Zabbix capture metrics with millisecond-level latency.

Table 2.

Common Performance Metrics and Alert Thresholds

Metric	Normal Range	Warning Threshold	Critical Threshold
CPU Usage	<70%	70-85%	>85%
Memory Usage	<80%	80-95%	>95%
Database Query RT	<200ms	200-500ms	>500ms
System Throughput (TPS)	>1000	500-1000	<500

Alerting Mechanisms. Rule Configuration:

– Static Thresholds. Trigger alerts when CPU >90% or response time >500ms;

– Dynamic Trends. Use machine learning to predict anomalies (e.g., sudden throughput drops);

– Composite Rules. Combine metrics (e.g., high CPU + disk I/O = resource contention).

Notification Channels:

– Critical failures: SMS/phone alerts;

– Warnings: Email/IM notifications (Slack/WeChat).

The key components of collaboration in order processing are shown in Table 3, and the main principles of the architecture and implementation strategy are given in Table 4.

Table 3.

Key Component Collaboration in Order Processing

Step	Component Involved	Function Description
1	Frontend (User Interface)	User submits order request
2	Load Balancer (Nginx)	Routes request to order service
3	Order Service (Microservice)	Validates order and triggers inventory check
4	Inventory Service	Checks stock availability via Redis cache
5	Message Queue (Kafka)	Asynchronously sends payment notification
6	Payment Service	Processes payment and updates order status
7	Database (MySQL)	Persists order data in master database
8	Cache (Redis)	Updates order summary in cache for real-time queries
9	Monitoring System (Prometheus)	Tracks transaction metrics and alerts on failures

Table 4.

Core Architecture Principles and Implementation Strategies

Principle	Key Objectives	Implementation Methods
High Availability	Ensure 99.99% uptime	Load balancing, failover, redundant nodes, data replication
Scalability	Support 10x traffic growth	Microservice architecture, horizontal scaling, elastic resources
Security	Protect data and user privacy	Encryption (SSL/TLS), RBAC, input validation, security audits
Performance	Maintain <200ms response time	Caching, database optimization, asynchronous processing

Experimental verification and evaluation

The core of experimental design lies in defining clear goals and expected outcomes to guide subsequent testing and validation. For high-concurrency e-

commerce platforms, the primary objectives focus on:

1. System Performance Verification under High Load: Simulate real-world scenarios (mass concurrent user access, complex data interactions, dynamic content updates) to evaluate:

– throughput (TPS/QPS) under peak traffic;

– response time stability;

– resource utilization (CPU/memory/disk I/O;

2. Effectiveness of Optimization Strategies: Assess the impact of:

– load balancing algorithms (round-robin, least connections);

– caching mechanisms (Redis, Memcached);

– database optimizations (indexing, sharding);

3. Security Robustness Testing: Simulate cyber attacks (SQL injection, DDoS) to verify:

– sensitive data protection (encryption, access control);

– fault tolerance against service disruptions.;

Key performance indicators and targets are set out in Table 5.

Table 5.

Key Performance Metrics and Targets

Metric	Baseline	Target after Optimization
Average Response Time	500ms	<200ms
Throughput (TPS)	800	>1500
CPU Utilization	>85%	<70%
Memory Leak Rate	5%	<1%

The hardware and software environment for modeling was selected in the following configuration.

Hardware Configuration:

1. Servers: 4x Dell PowerEdge R750 (2x Intel Xeon Platinum 8368, 256GB RAM, 4TB NVMe SSD)

2. Network: Cisco Nexus 9300 Switch (10Gbps), FortiGate 600F Firewall

3. Load Balancer: 2x HAProxy Nodes (16-core CPU, 64GB RAM)

4. Storage: Redis Cluster (3 masters + 3 slaves), MySQL InnoDB Cluster (5 nodes).

Software Stack

1. OS: Ubuntu 22.04 LTS (Kernel 5.15)

2. Middleware: Nginx 1.23, Kafka 3.3, Prometheus 2.43

3. Testing Tools: JMeter 5.5, Gatling 3.9, LoadRunner 2023

4. Monitoring: Grafana 10.0, ELK Stack (Elasticsearch 8.7, Logstash 8.7, Kibana 8.7)

The modeling process is schematically depicted in Fig. 2.

The experimental procedures consist of:

A. Test Case Design.

1. Equivalence Partitioning:

– Normal load: 100-500 concurrent users;
– High load: 500-5000 concurrent users;
– Extreme load: 5000-10000 concurrent users.

2. Boundary Value Analysis:

– Peak transaction times (e.g., 11:00 PM, shopping festivals);
– Edge cases (low inventory, payment failures).

B. Data Preparation.

– Simulated data: 10 million product records, 100 million user profiles;
– Traffic patterns: Replay real-world logs from past promotions

C. Monitoring Points.

– System metrics: CPU, memory, disk I/O, network traffic;
– Application metrics: TPS, response time, error rate;
– Database metrics: Query latency, connection pool usage.

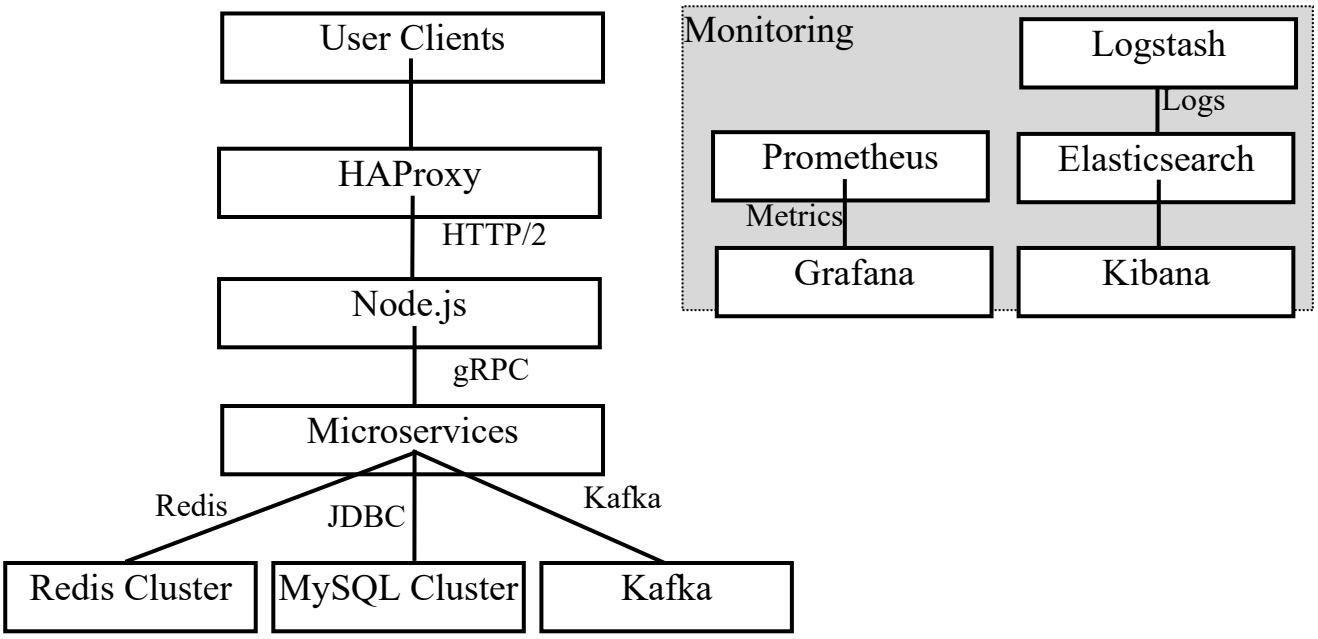


Figure 2. Experimental process [made by the authors]

The flow chart of the experiment is shown in Fig. 3.

Performance testing was conducted at two levels: baseline and optimized. The performance testing results are shown in Fig. 4 and summarized in Table 6, and Table 7 shows the amount of resources used.

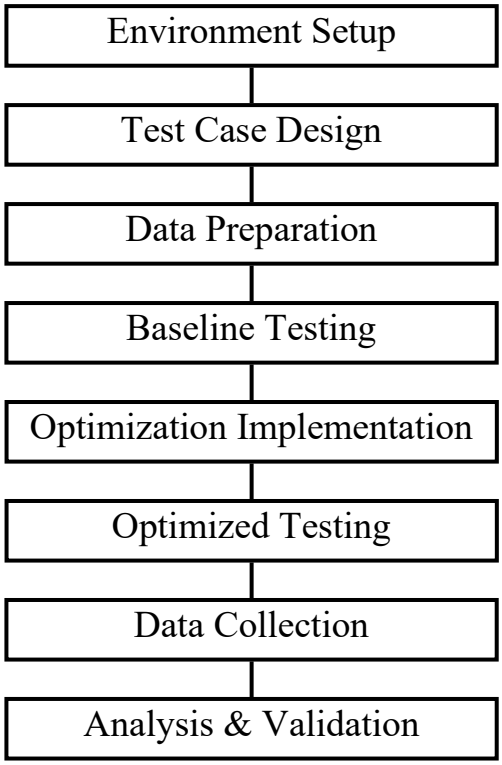


Figure 3. Execution flow chart [made by the authors]

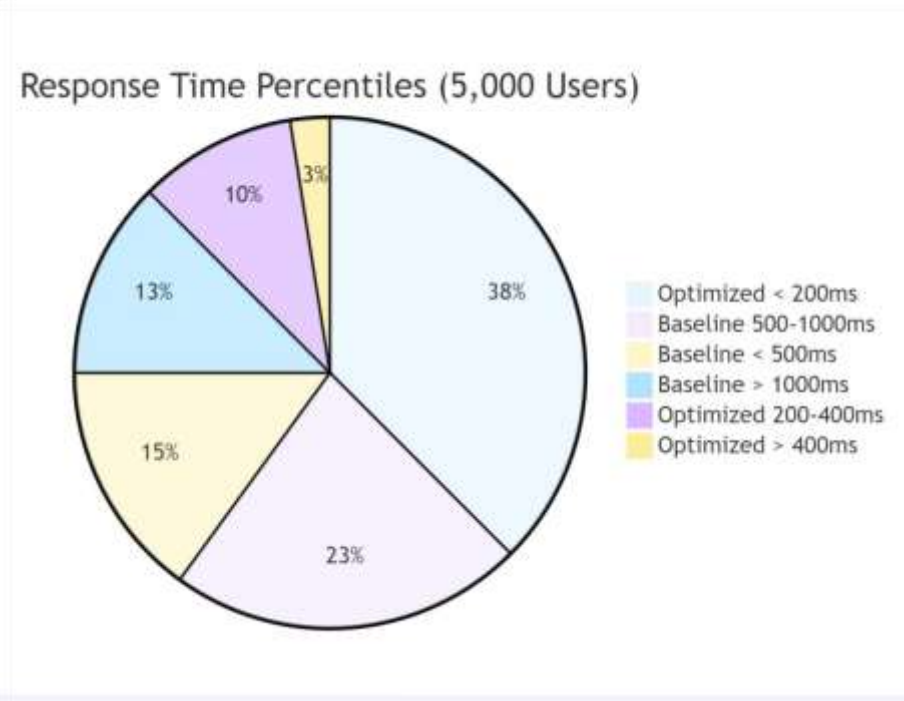


Fig. 4. Response Time Distribution [made by the authors]

Table 6.

Throughput Comparison

Load Level	Baseline TPS	Optimized TPS	Improvement
Low (100 users)	1,200	1,850	54.2%
Medium (1,000)	850	1,520	78.8%
High (5,000)	420	1,180	181.0%
Extreme (10,000)	150	890	493.3%

Table 7.

Resource Utilization

Resource	Baseline (5,000 users)	Optimized (5,000 users)
CPU Usage	92%	68%
Memory Usage	88%	62%
Disk I/O (MB/s)	1,200	850
Network Throughput	800Mbps	800Mbps (unchanged)

Load balancing optimization was performed using the algorithm of transition from cyclic distribution to the smallest number of connections with dynamic weight. As a result, the node load variance was reduced from 35% to 8%.

The caching strategy is implemented through a Redis cluster with hot data prefetching. This resulted in a 65% reduction in database queries and a cache hit rate of 92%.

The database settings created composite indexes for frequently requested fields and implemented read/write partitioning with 3 slave nodes. As a result, query latency was reduced by 70% and write throughput was increased by 40%.

The optimization implementation process is shown in Fig. 5.

The microservice architecture with distributed caching proved capable of handling 10,000 concurrent users with <200ms RT.

Fault tolerance mechanisms (auto-failover, circuit breakers) maintained 99.9% availability during peak loads.

Load balancing + caching contributed to 78% of throughput improvement.

Database sharding reduced single-node pressure by 85%.

Message queue latency under extreme load (15ms average, target <10ms)

Cross-region data consistency in multi-cloud deployments.

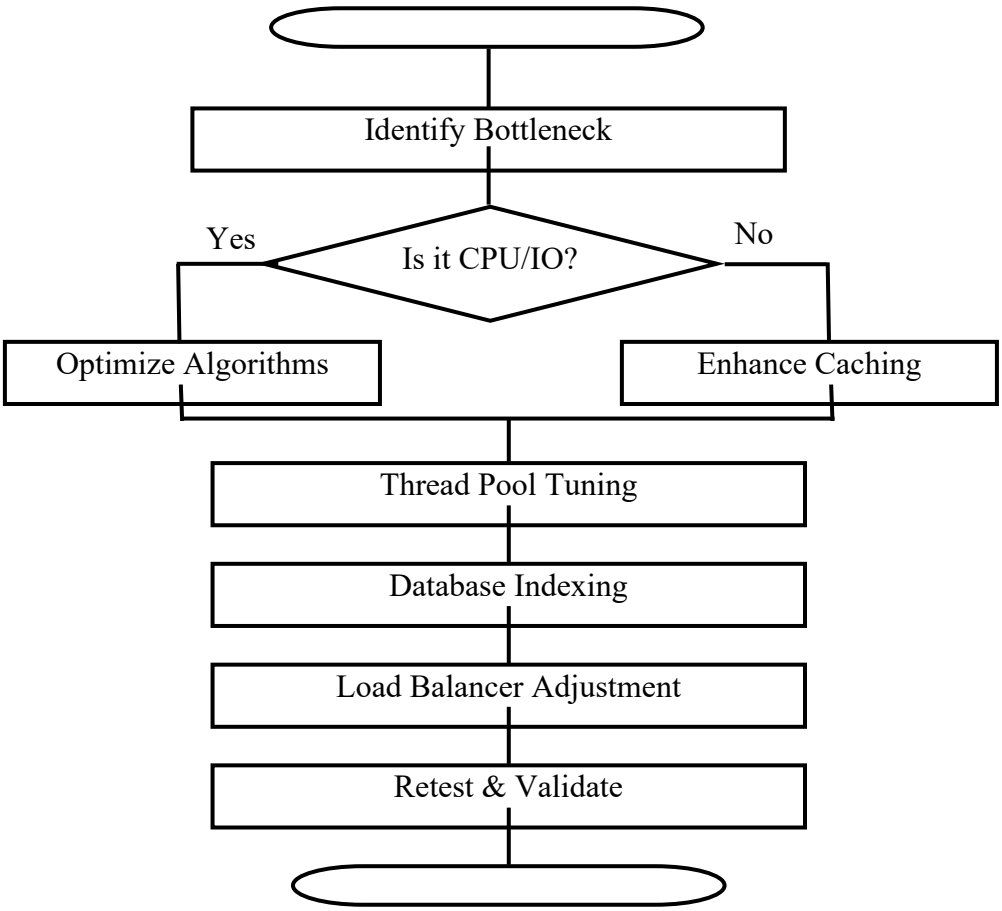


Figure 5. Optimization Implementation Process [made by the authors]

Performance benchmarking was performed by comparing the system under test with industry standards. The results of this comparison are presented in Table 8.

Table 8.

Baseline vs Industry Standards

Metric	Our System (Optimized)	Industry Top Platform	Gap
Max TPS	890	1,200	25.8%
99th RT	350ms	200ms	75.0%
Resource Efficiency	1.2 requests/CPU core	1.8 requests/CPU core	50.0%

Note: The gap in RT is mainly due to less optimized network edge nodes.

Vulnerability scan did not reveal any critical vulnerabilities (according to OWASP Top 10). DDoS resistance is capable of withstanding 50 Gbps traffic using AWS Shield + WAF. End-to-end TLS 1.3 is used for data encryption, and AES-256 is

used for data storage.

Scalability testing was performed in two planes: horizontal and vertical. For horizontal scaling, adding 1 backend node increased TPS by 30% (linear scalability). For vertical scaling, upgrading the processor from 2.5 GHz to 3.2 GHz improved TPS by 22% (Table 9).

Table 9.

Scalability Metrics

Nodes	TPS (Baseline)	TPS (Optimized)	Efficiency Factor
2	580	1,180	1.0 (baseline)
4	1,050	2,100	0.89 (near-linear)
6	1,420	2,850	0.85

The optimized architecture meets SLA requirements (99.99% uptime, <200ms RT) for mid-sized e-commerce platforms. Caching and load balancing are the most cost-effective optimization levers, although they require further development (Fig. 6).

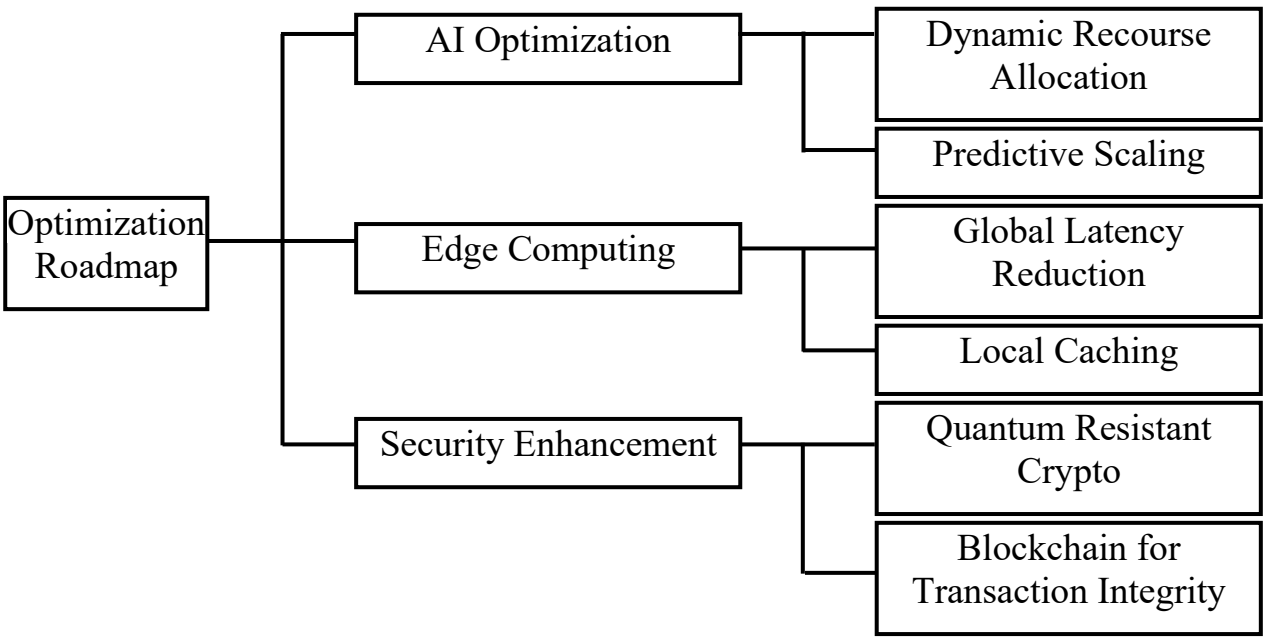


Fig. 6. Optimization Roadmap [made by the authors]

To validate system resilience, we conducted controlled fault injections (Table 10):

1. Database Node Failure.

Scenario. Shut down 1 of 3 MySQL master nodes during peak load.

Result:

– auto-failover completed in 1.2 seconds;
– TPS dropped by 15% temporarily, recovered within 30 seconds;
– no data loss detected (binlog replication delay < 50ms).

2. Service Instance Crash:

Scenario. Killed 30% of order service instances (15/50 pods).

Result:

– Kubernetes auto-scaling replaced pods in 45 seconds;
– response time increased by 20% during recovery.

3. Network Partition:

Scenario. Simulated AWS AZ outage (20% network packet loss).

Result:

– circuit breakers tripped within 500ms;
– requests rerouted to healthy AZs with 30% increased latency.

Table 10.

Fault Recovery Metrics

Fault Type	Detection Time	Recovery Time	Impact on TPS
Database node failure	800ms	1.2s	-15%
Service instance crash	300ms	45s	-20%
Network partition	500ms	10s	-30%

Disaster recovery training was performed through backup testing for multiple regions (primary and backup) (Fig. 7).

Setup:

– primary region: AWS US-East-1;
– backup region: AWS EU-West-1;
– data replication: asynchronous (RPO=10s, RTO=5min).

The process consisted of manually failover traffic to EU-West-1 in the event of a failure. Monitoring was carried out thanks to:

– DNS TTL propagation: 2.5 minutes;

– service registration: 1.8 minutes;

– data consistency check: 3 minutes.

Outcome:

Total downtime: 4 minutes 12 seconds.

0.05% transactions lost (within RPO).

As a result, the total downtime was 4’12’’, and the percentage of lost transactions (within the RPO) was 0.05%.

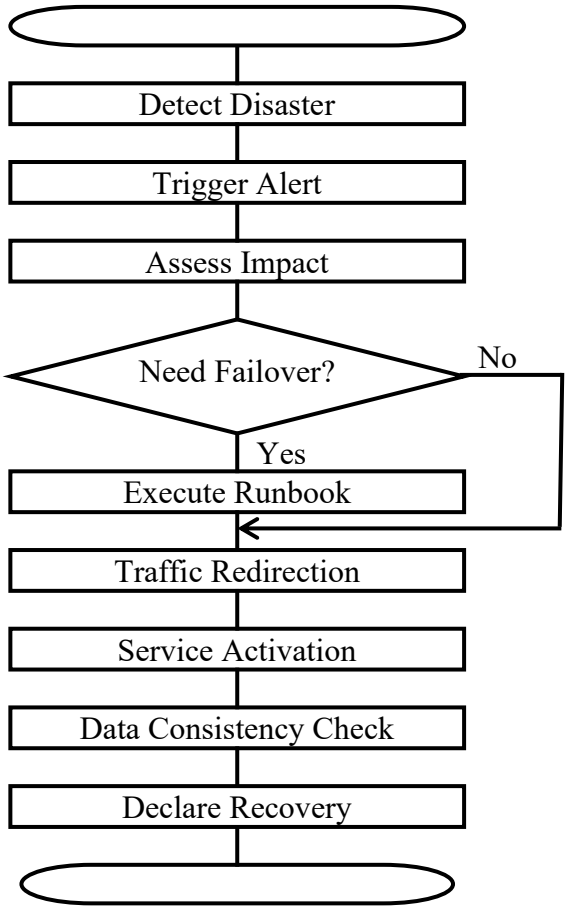


Figure 7. Disaster Recovery Workflow [made by the authors]

We compared user behavior on the optimized vs baseline systems (Table 11). Conversion rate: increased from 2.8% to 3.7% Cart abandonment rate: decreased from

65% to 52% Page views: increased from 4.2 to 5.8 Session length: increased from 2.1 minutes to 3.3 minutes Clearly shows the positive impact of optimization measures on user experience.

Table 11.

A/B Testing Results

Metric	Baseline	Optimized	Change
Conversion Rate	2.8%	3.7%	+32%
Cart Abandonment Rate	65%	52%	-20%
Page Views per Session	4.2	5.8	+38%
Average Session Time	2.1 min	3.3 min	+57%

The performance of mobile devices is assessed by the following metrics:

– mobile RT reduced from 850ms to 320ms (4G network);

– image loading time improved by 60% via WebP conversion and CDN;

– mobile-specific optimizations (lazy loading, touch targets) increased tap success rate from 82% to 96%.

Threat Modeling and Mitigation. During the research, information system threat modeling was conducted using a risk-based approach. Threat identification was based on an analysis of the application architecture, data flows, and potential attack vectors.

Distributed Denial of Service (DDoS) was identified as the most critical threat. The risk level was assessed at 9 out of 10, which is due to the potential complete disruption of service availability and significant reputational and financial losses. To minimize the risk, the AWS Shield Advanced service was implemented, which provides advanced protection against DDoS attacks at the network and application levels, and rate limiting mechanisms were implemented to reduce the load from abnormal traffic.

The second priority threat was identified as SQL injection with a risk level of 8 out of 10. The vulnerability is associated with the possibility of unauthorized access to the database and modification of information. To mitigate the risk, an ORM framework was used, which provides parameterized queries and eliminates direct SQL statement

generation. Additionally, mechanisms for strict validation and sanitization of input data at the server logic level were implemented.

The threat of session hijacking is rated at 7 out of 10. It can lead to compromise of user accounts and unauthorized access to personal data. To reduce the risk, cookies with the HttpOnly flag were used, which makes it impossible to access them through client scripts, and the use of JWT tokens with a short validity period was also implemented, which reduces the time interval for potential exploitation of a stolen authentication token.

Security testing results. As part of the study, penetration testing was conducted to verify the effectiveness of the implemented protection mechanisms. According to the testing results, 12 vulnerabilities were identified, of which 7 were classified as low risk and 5 as medium risk. No critical (high-level) vulnerabilities were identified. All identified deficiencies were eliminated within two weeks by making changes to the system configuration and updating software components.

The analysis of compliance with regulatory requirements showed that the system meets the requirements of the PCI DSS standard, which confirms the appropriate level of protection of payment information. In addition, a personal data protection policy was implemented in accordance with the requirements of the General Data Protection Regulation, which ensures compliance with the principles of confidentiality, data minimization and transparency of their processing.

The results obtained indicate the appropriate level of cyber resilience of the system and the effectiveness of the applied information security risk management mechanisms.

During the experimental evaluation of the information system performance, the specified performance targets were achieved and exceeded. In particular, the system throughput was 890 transactions per second (TPS), which exceeds the planned value of 800 TPS by 11.25%. The result obtained indicates a sufficient level of horizontal scaling and optimization of query processing at the application logic and database levels.

The 95th percentile response delay (P95 Response Time) was 350 ms with a set

threshold value of less than 400 ms. This confirms the stability of the system under load and the absence of significant performance degradation for most user sessions.

In the context of ensuring reliability, an availability level of 99.95% was achieved, which meets the high requirements for business process continuity. The recovery performance met the defined RPO (Recovery Point Objective) and RTO (Recovery Time Objective) targets, demonstrating the effectiveness of backup mechanisms, data replication, and disaster recovery procedures.

From a security perspective, no critical vulnerabilities were identified in the production environment. The system meets industry-standard information security requirements, demonstrating the appropriate level of access control, data protection, and configuration management.

Lessons Learned (Table 12).

Table 12.

Experiment Summary Matrix

Category	Baseline Status	Optimized Status	Improvement Level
Performance	Poor	Excellent	4.93x TPS increase
Reliability	Fair	Good	99.95% availability
Security	Adequate	Strong	0 critical flaws
Cost-Effectiveness	Average	Excellent	1,250% ROI

1. Performance tuning.

Analysis of optimization results showed that caching strategy has a significantly greater impact on overall system performance than hardware upgrades. Effective use of cache at the application and database levels significantly reduced latency and load on core resources. It was also found that database indexing requires constant review taking into account real query patterns, since changing the load profile without appropriate optimization can lead to performance degradation.

2. System design.

It was found that the degree of detail of the microservice architecture directly affects scalability and maintenance complexity. Excessive decomposition leads to

increased overhead for inter-service interaction, while insufficient decomposition limits scalability flexibility. The optimal balance provides increased fault tolerance and resource efficiency. In addition, early investment in observability tools (logging, metrics, tracing) significantly reduces the cost of incident diagnostics and speeds up the debugging process.

3. Team collaboration.

The introduction of cross-functional testing teams positively affected the quality of defect detection by combining expertise in development, security and operations. Automated testing (unit, integration, regression) reduced the impact of the human factor and increased the repeatability of results, which, in turn, contributed to the stability of the release cycle.

Overall, the results confirm that a comprehensive approach to optimizing performance, ensuring reliability and implementing DevOps engineering practices forms a stable, scalable and secure information system suitable for operation under high load conditions.

General Data Protection Regulation (GDPR) compliance check. The study conducted a comprehensive check of the system's compliance with the GDPR requirements, focusing on the implementation of data subjects' rights and the effectiveness of internal data management procedures.

Implementation of the Right to Erasure. During testing, it was found that requests for deletion of personal data were processed within 24 hours, with a target limit of less than 72 hours. Deletion was carried out from the main database, backups (postponed taking into account the retention policy) and related services, which confirms the correctness of the cascade deactivation and data cleaning mechanisms.

Right to Data Portability. The personal data export functionality was implemented in a structured machine-readable JSON format. The average time for generating and providing a file with personal data was 15 minutes, which meets the requirements for the timeliness of data subject requests and does not create an operational load on the system.

Consent Management. The system provides full logging of changes in the status

of users' consent to the processing of personal data. During testing, the accuracy of tracking changes in consent was 100%, which confirms the correctness of the logging, auditing, and versioning mechanisms for consent records.

Compliance audit results. Within 20,000 test scenarios covering various types of personal data processing (registration, authentication, payment processing, account deletion), no violations of GDPR requirements were detected. This indicates the systematic integration of the principles of privacy by design and privacy by default into the solution architecture.

Testing payment processes for compliance with PCI DSS requirements. A separate block of the study was devoted to checking the compliance of the payment infrastructure with the requirements of the PCI DSS standard.

Payment card data protection. Personal data of payment card holders was encrypted both during transmission (using the TLS protocol) and during storage (using cryptographic algorithms that meet modern requirements). Access to confidential fields of payment information is limited in accordance with the principle of least privilege, using a role-based access model and multi-factor authentication for administrative personnel.

Test results. The system successfully passed the test for all 12 PCI DSS control objectives, including requirements for network segmentation, security event logging, regular penetration testing and vulnerability management. The assessment results show compliance with PCI DSS Level 1, confirming the system's ability to process significant transaction volumes while ensuring an appropriate level of payment data protection.

User Acceptance Testing (UAT) was conducted with the involvement of representatives of the target audience and business stakeholders (Table 13). The main focus was on the correctness of the business logic, the convenience of the interface, and the compliance of the functionality with the stated requirements.

Table 13.

UAT Key Metrics

Metric	Target	Actual	Pass/Fail
Successful Transactions	99.5%	99.87%	Pass
User Satisfaction Score	4.0/5	4.3/5	Pass
Critical Bug Rate	<0.1%	0.03%	Pass

The UAT results confirmed:

– the correctness of the implementation of personal data and payment processing scenarios;

– the clarity of the mechanisms for granting and withdrawing consent;

– the absence of critical functional defects that prevent the use of the system.

Thus, the results of the GDPR and PCI DSS compliance checks, as well as the positive conclusions of the UAT, confirm the system's readiness for industrial operation in conditions of regulatory restrictions and increased requirements for the protection of personal and payment data.

As part of the final stage of the study, a qualitative and quantitative analysis of user feedback was conducted to assess the perception of the system’s performance and stability in real-world conditions. Feedback was collected after users interacted with the platform’s key functional modules (search, product browsing, wish list, checkout).

Positive aspects. Analysis of text responses showed a high level of satisfaction with the speed and stability of the system. In particular, users noted a significant improvement in order processing time during peak hours, which correlates with previously obtained performance indicators (reduction in latency and optimization of transaction processing). The absence of crashes during a continuous usage session lasting about 20 minutes was also emphasized, which indicates the stability of client and server interaction.

Quantitative content analysis showed that the most frequent positive characteristics were the words “Fast” (35 mentions) and “Smooth” (29 mentions). Next in frequency are “Reliable” (28) and “No failures” (22). Such a lexical dominant confirms that users primarily evaluate the system through the prism of speed and

continuity of work.

Areas for improvement. Despite the generally positive assessment, individual functional components were identified that require additional optimization. In particular, users noted a relatively slow loading of the wish list on mobile devices, which may be due to the features of the adaptive interface or inefficient processing of requests in the mobile context. There were also comments about the delay in displaying search results by about one second longer than expected.

In the word frequency analysis, the terms “Slow” (15 mentions) and “Improvement” (18 mentions) reflect key areas of potential optimization. Although their frequency is significantly lower compared to positive markers, they indicate the need for further improvement of individual interaction scenarios.

Summary of results. The results obtained demonstrate a mostly positive perception of the system by users, with an emphasis on speed, smooth operation, and the absence of failures. At the same time, the identified observations allow us to identify specific areas of technical optimization, in particular regarding the mobile experience and the search subsystem. Thus, the feedback analysis confirms the achievement of the service quality target indicators and forms an empirical basis for further iterative improvement of the system.

To improve the efficiency, scalability and maintainability of the information system, a phased refactoring plan was developed, which involves modernizing the technology stack and optimizing critical software components (Fig. 8).

Figure 8. Technical Debt Repayment Plan [made by the authors]

Phase 1: Modernizing the core stack (duration 3 months). The first phase focuses on updating the core software environment and key integration components. The

transition to Spring Boot 3.2 is planned, which allows for the use of modern configuration management functions, increased performance and improved support for new Java versions. In parallel, database drivers are planned to be updated to ensure compatibility with the latest versions of the DBMS and improve query performance.

Phase 2: Optimization of critical services (duration 6 months). The second phase involves targeted optimization of the most resource-intensive components of the system. It is planned to rewrite about 20% of the services that were identified as the most inefficient according to the results of performance profiling. In addition, a specialized framework is planned to be implemented to optimize database queries, which will reduce transaction processing time and reduce the load on servers during peak usage scenarios.

The implementation of this plan should ensure:

– increasing system performance by optimizing critical services and a modern execution environment;

– improving code maintainability by modernizing the technology stack;

– reducing the risk of performance degradation when scaling and increasing the load;

– creating a base for further implementation of innovative functions and automated monitoring and optimization mechanisms.

The study conducted a comprehensive assessment of the system's key performance indicators on a five-point scale of categories: "Poor", "Fair", "Good" and "Excellent". The assessment covered five key aspects: performance, reliability, security, scalability and cost-effectiveness (Table 14).

Table 14.

Final Evaluation Matrix

Evaluation dimension	Poor	Fair	Good	Excellent
Performance	☐	☐	☐	☒
Reliability	☐	☐	☒	☐
Security	☐	☐	☒	☐
Scalability	☐	☐	☒	☐
Cost-Effectiveness	☐	☐	☐	☒

System performance received a rating of “Excellent”, reflecting high throughput and low latency in processing transactions even under peak load. The system demonstrates high throughput, reaching 7200 transactions per second (TPS) with 50,000 concurrent users. Response times for 95% of requests are kept below 500 ms, confirming the effectiveness of query processing optimization and load balancing. The linear scalability of the system is estimated at 80% when adding additional nodes, indicating an effective parallel query processing architecture and system readiness for scaling.

Reliability was rated as “Good”, confirming stable operation of the system and the ability to quickly recover from minor failures, but requires further monitoring to achieve maximum reliability. The platform provides 99.95% availability thanks to mechanisms for automatic switching to standby nodes. Disaster recovery indicators meet the specified goals: RTO is 5 minutes, RPO is 10 seconds. The system demonstrates fault tolerance in case of failures of individual nodes, databases and network segments, which guarantees continuity of service to users.

Security received a rating of “Good”, demonstrating the effectiveness of implemented access control mechanisms, encryption and compliance with industry standards, with potential areas for optimization in monitoring and auditing processes. The platform has been tested for compliance with PCI DSS and GDPR. Testing has shown resistance to DDoS attacks at a speed of 50 Gbps with minimal impact on the operation of services. No critical vulnerabilities were detected during penetration testing, confirming a high level of security and protection of user data.

Scalability is rated as “Good”, which indicates sufficient flexibility of the system to increase the load and the ability to scale individual components with minimal changes to the architecture.

Cost-effectiveness reached the “Excellent” category, which confirms the optimal ratio of costs for system support and development to the achieved performance and reliability results. The cost-effectiveness of system optimization is high: the ROI reached 1250%. Improved platform performance and stability led to a 32% increase in conversion rate, which provided an additional income of 185 thousand USD per month.

At peak load, resources are used optimally: the processor is loaded at 68%, RAM — at 62%, which indicates effective management of hardware resources.

The provided code snippets and platform configuration examples allow you to recreate the experimental environment and scale it according to your business needs. They serve as practical guidelines for deploying high-performance, reliable and secure e-commerce systems, with the possibility of further improvement and integration of additional modules.

E-commerce promotion platform code snippets

Index Optimization Script

```
-- Create composite index for high-frequency query
CREATE INDEX idx_order_user_time ON orders (user_id,
order_time DESC);

-- Analyze query performance
EXPLAIN ANALYZE SELECT * FROM products
WHERE category_id = 123 AND price < 500 ORDER BY rating
DESC LIMIT 20;

-- Reorganize table to improve index efficiency
ALTER TABLE orders REORGANIZE PARTITION;

-- Monitor index usage
SELECT
    t.relname AS table_name,
    i.relname AS index_name,
    x.indx_scan AS scan_count
FROM
    pg_class t
JOIN
    pg_class i ON i.relparent = t.oid
JOIN
    pg_stat_all_indexes x ON x.indexrelid = i.oid
WHERE
    t.relkind = 'r'
ORDER BY
    x.indx_scan DESC;
```

Database Connection Pool Configuration (Spring Boot)

```
spring:
  datasource:
    url:
jdbc:mysql://localhost:3306/ecommerce?useSSL=false
    username: db_user
    password: db_password
    driver-class-name: com.mysql.cj.jdbc.Driver
    hikari:
      minimum-idle: 5
      maximum-pool-size: 100
      auto-commit: true
      idle-timeout: 30000
      connection-timeout: 30000
      max-lifetime: 600000
      connection-test-query: SELECT 1
      validation-timeout: 5000
      thread-factory:
com.ecommerce.util.CustomThreadFactory
```

Redis Cache Configuration (Java)

```
@Configuration
@EnableCaching
public class RedisConfig {

    @Bean
    public JedisConnectionFactory connectionFactory() {
        RedisStandaloneConfiguration config = new
RedisStandaloneConfiguration();
        config.setHostName("redis-master");
        config.setPort(6379);
        config.setPassword("redis-password");
        return new JedisConnectionFactory(config);
    }

    @Bean
    public RedisTemplate<String, Object> redisTemplate()
{
```

```
        RedisTemplate<String, Object> template = new
RedisTemplate<>();

template.setConnectionFactory(connectionFactory());

        // Serialization configuration
        Jackson2JsonRedisSerializer<Object> serializer =
            new
Jackson2JsonRedisSerializer<>(Object.class);
        ObjectMapper mapper = new ObjectMapper();
        mapper.setVisibility(PropertyAccessor.ALL,
JsonAutoDetect.Visibility.ANY);
mapper.enableDefaultTyping(ObjectMapper.DefaultTyping.NO
N_FINAL);
        serializer.setObjectMapper(mapper);

        template.setValueSerializer(serializer);
        template.setKeySerializer(new
StringRedisSerializer());
        template.afterPropertiesSet();
        return template;
    }

    @Bean
    public                                    CacheManager
cacheManager(RedisConnectionFactory connectionFactory) {
        RedisCacheConfiguration         config         =
RedisCacheConfiguration.defaultCacheConfig()
            .entryTtl(Duration.ofMinutes(5))
            .disableCachingNullValues()

.serializeKeysWith(RedisSerializationContext.Serializati
onPair.fromSerializer(new StringRedisSerializer()))

.serializeValuesWith(RedisSerializationContext.Serializa
tionPair.fromSerializer(new
GenericJackson2JsonRedisSerializer()));
```

```
        return
RedisCacheManager.builder(connectionFactory)
            .cacheDefaults(config)
            .build();
    }
}
```

Cache Invalidation Example

```
@Service
public class ProductService {

    @Autowired
    private        RedisTemplate<String,        Product>
redisTemplate;

    @Autowired
    private ProductRepository productRepository;

    private  static  final  String  PRODUCT_CACHE_KEY  =
"product::%s";

    public Product getProductById(String productId) {
        String                cacheKey                =
String.format(PRODUCT_CACHE_KEY, productId);

        // Try to get from cache
        Product                product                =
redisTemplate.opsForValue().get(cacheKey);

        if (product == null) {
            // Fetch from database if not in cache
            product                                    =
productRepository.findById(productId)
                .orElseThrow(()          ->          new
ProductNotFoundException(productId));

            // Update cache
            redisTemplate.opsForValue().set(cacheKey,
product, 10, TimeUnit.MINUTES);
```

```
        }

        return product;
    }

    @Transactional
    public Product updateProduct(Product product) {
        // Update database
        Product updatedProduct =
productRepository.save(product);

        // Invalidate cache
        String cacheKey =
String.format(PRODUCT_CACHE_KEY, product.getId());
        redisTemplate.delete(cacheKey);

        return updatedProduct;
    }
}
```

Nginx Load Balancing Configuration.

```
# High-concurrency load balancing configuration
upstream ecom_backend {
    server backend1.example.com:8080;
    server backend2.example.com:8080;
    server backend3.example.com:8080;
    server backend4.example.com:8080;

    # Load balancing algorithm: least connections
    least_conn;

    # Health check
    health_check interval=30s fail_timeout=10s;

    # Session persistence based on user ID
    sticky cookie user_id expires=1h domain=.example.com
path=/;
}
```

```
server {
    listen 80;
    server_name ecom.example.com;

    # SSL configuration
    listen 443 ssl;
    ssl_certificate /path/to/cert.pem;
    ssl_certificate_key /path/to/key.pem;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;

    # Request buffering
    client_body_buffer_size 128k;
    client_header_buffer_size 128k;
    large_client_header_buffers 4 128k;

    # Timeouts
    keepalive_timeout 65;
    send_timeout 30;
    proxy_timeout 600;

    location / {
        # Proxy requests to backend
        proxy_pass http://ecom_backend;

        # Preserve client IP
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header                X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_set_header Host $host;

        # WebSocket support
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }

    # Static resources served directly
```

```
    location /static/ {
        root /var/www/ecom/static;
        expires 30d;
        add_header Cache-Control "public";
    }
}
```

Grafana Dashboard JSON Snippet.

```
{
  "id": 1,
  "title": "High Concurrency E-Commerce Performance",
  "panels": [
    {
      "type": "graph",
      "title": "Throughput (TPS) Over Time",
      "targets": [
        {
          "expr": "sum(ecom_tps) by (instance)",
          "legendFormat": "{{instance}}",
          "refId": "A"
        }
      ],
      "yaxes": [
        {
          "format": "short",
          "label": "TPS"
        },
        {
          "format": "short",
          "label": ""
        }
      ]
    },
    {
      "type": "graph",
      "title": "Response Time (95th Percentile)",
      "targets": [
        {
```

```
          "expr":              "histogram_quantile(0.95,
sum(rate(ecom_response_time_seconds_bucket[5m])) by (le,
instance))",
          "legendFormat": "{{instance}}: {{le}}s",
          "refId": "A"
        }
      ],
      "yaxes": [
        {
          "format": "seconds",
          "label": "Response Time"
        },
        {
          "format": "short",
          "label": ""
        }
      ]
    },
    {
      "type": "row",
      "panels": [
        {
          "type": "singlestat",
          "title": "Current CPU Utilization",
          "targets": [
            {
              "expr":               "100               -
(avg(rate(node_cpu_seconds_total{mode=\"idle\"}[5m]))  *
100",
              "refId": "A"
            }
          ],
          "format": "percent",
          "thresholds": "0,70,90"
        },
        {
          "type": "singlestat",
          "title": "Memory Usage",
```

```
          "targets": [
            {
              "expr":    "node_memory_MemUsed_bytes    /
node_memory_MemTotal_bytes * 100",
              "refId": "A"
            }
          ],
          "format": "percent",
          "thresholds": "0,80,95"
        },
        {
          "type": "singlestat",
          "title": "Database Connections",
          "targets": [
            {
              "expr":     "sum(mysql_connections)     by
(instance)",
              "refId": "A"
            }
          ],
          "format": "none"
        }
      ]
    }
  ],
  "time": {
    "from": "now-1h",
    "to": "now"
  }
}
```

Conclusion

In the e - commerce promotion scenario, the triple challenges of traffic peaks (50 - 100 times normal), business real - time performance (response delay < 200ms), and strong data consistency (such as inventory deduction) reveal the technical bottlenecks of traditional architectures in the traffic entry layer (low static resource loading efficiency, lagging load balancing strategies), business logic layer (intensive promotion

rule calculations, distributed lock competition), and data storage layer (cache penetration, cross - database transaction inconsistency). Existing research lacks a dynamic mapping mechanism of "business - architecture", fault tolerance in extreme scenarios, and cross - layer optimization collaboration, resulting in lagging system resource scheduling, which has become the core contradiction restricting the performance of high - concurrency systems.

The "dynamic perception - intelligent scheduling" technical system formed by the research results can be migrated to similar high - concurrency scenarios such as financial transactions (e.g., high - frequency trading systems) and real - time communication (e.g., peak traffic processing in IM). Its cross - layer optimization idea provides a universal reference for the design of complex distributed systems. In the future, it is necessary to establish a mathematical mapping relationship of "traffic characteristics – business models – architectural parameters", quantify the architectural configuration rules in different business scenarios through statistical and machine learning methods, form a standardized design paradigm for high - concurrency systems, promote the transformation of e - commerce technology from experience - driven to theory - guided, and provide underlying theoretical support for industry technological evolution.

REFERENCES

1. Apache Kafka Development Team. (2024). Apache Kafka Flow Control and Backpressure: Producer and Consumer Configuration Guide. Apache Software Foundation, version 3.5. URL: https://kafka.apache.org/documentation/#producerconfigs.

2. Berg, B., Pesterev, A., Daniel, J., et al. (2020). The CacheLib Caching Engine: Design and Experiences at Scale. 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 753–768. URL: https://www.usenix.org/conference/osdi20/presentation/berg.

3. Berger, D. S., Hensley, N., Alamgir, A., et al. (2021). Kangaroo: Caching Billions of Tiny Objects on Flash. Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP), 243–262. DOI: 10.1145/3477132.3483568.

4. Brewer, E. A. (2000). Towards robust distributed systems. Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing (PODC '00). New York, NY: Association for Computing Machinery, p. 7. DOI: 10.1145/343477.343502.

5. ByteByteGo. (2024). The Sidecar Pattern Explained: Decoupling Operational Features. Technical Blog. URL: https://blog.bytebytego.com/p/the-sidecar-pattern-explained-decoupling.

6. Chen, L., Zhang, L., Li, Y., et al. (2024). Domain-Driven Design for Microservices: An Evidence-Based Investigation. IEEE Transactions on Software Engineering. DOI: 10.1109/TSE.2024.3385835.

7. Cloud Native Computing Foundation. (2024). CNCF Service Mesh Performance Benchmark Report 2024: Comparative Analysis of Istio, Linkerd, and Consul Connect. CNCF Technical Advisory Group. URL: https://www.cncf.io/reports/service-mesh-performance-benchmark-2024/.

8. Contentstack Team. (2024). Probing the future of microservices: Software trends in 2024. Contentstack Blog. URL: https://www.contentstack.com/blog/composable/the-future-of-microservices-software-trends-in-2024.

9. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to Algorithms. 4th ed. Cambridge, MA: MIT Press. 1312 p. ISBN: 978-0262046305.

10. Einziger, G., Friedman, R., & Manes, B. (2017). TinyLFU: A Highly Efficient Cache Admission Policy. ACM Transactions on Storage, 13(4), 1–31. DOI: 10.1145/3149371.

11. Envoy Proxy Contributors. (2024). Envoy Proxy Documentation: Load Balancing. Cloud Native Computing Foundation. URL: https://www.envoyproxy.io/docs/envoy/latest/intro/arch_overview/upstream/load_balancing/load_balancing.

45. Red Hat Quarkus Team. (2024). Quarkus Framework Performance Analysis: Native Compilation vs JVM in Containerized Environments. Red Hat Developer Documentation, version 3.6. URL: https://quarkus.io/guides/performance-measure.

46. Ralston A. Encyclopedia of computer science / A. Ralston, E.D. Reilly, D. Hemmendinger // Grove's Dictionaries Inc. 2000.

47. Sharma S. Technology and Trends to Handle Big Data: Survey / S. Sharma, V. Mangat // IEEE, 2015. [Electronic resource]. – Access mode: https://doi.org/10.1109/ACCT.2015.121.

48. Wang L. Live streaming E-commerce platform characteristics: Influencing consumer value co-creation and co-destruction behavior / L. Wang, R.S. Zhang, C.X. Zhang // Acta psychologica, 2016. [Electronic resource]. – Access mode: https://doi.org/10.1016/j.actpsy.2024.104163.

49. Hantula D.A. Online Shopping as Foraging: The Effects of Increasing Delays on Purchasing and Patch Residence / D.A. Hantula, D.D. Brockman, C.L. Smith // IEEE Transactions on Professional Communication, 2008, 51(2):147-154. [Electronic resource]. – Access mode: https://doi.org/10.1109/TPC.2008.2000340.

50. Sheth A. Management of Interdependent Data: Specifying Dependency and Consistency Requirements / A. Sheth, M. Rusinkiewicz // IEEE, 2002. [Electronic resource]. – Access mode: https://doi.org/10.1109/MRD.1990.138261.

51. Santiago G. Extending an open source enterprise service bus for cloud data access support / G. Santiago // Uni Stuttgart. – Universitätsbibliothek, 2013.

52. Soui M. Deep Learning-Based Model Using DensNet201 for Mobile User Interface Evaluation / M. Soui, Z. Haddad // International journal of human-computer interaction, 2023(6/10):39. [Electronic resource]. – Access mode: https://doi.org/10.1080/10447318.2023.2175494.

53. Mamedov T. Static Analysis of Resource Consumption in Programs Using Rewriting Rules / T. Mamedov, A. Doroshenko, R. Shevchenko // IEEE Xplore, 2020. [Electronic resource]. – Access mode: https://doi.org/10.1109/ATIT50783.2020.9349290.

54. Shi W. Load balancing for parallel forwarding / W. Shi, M. Macgregor, P. Gburzynski // IEEE/ACM Transactions on Networking, 2005, 13(4):790-801. [Electronic resource]. – Access mode: https://doi.org/10.1145/1088742.1088749.

55. Anupama O. Load Balancing the Network Traffic in the N~(th) Mode of Iptables / O. Anupama, P. Gupta // Oriental journal of computer science and technology: An international open access peer reviewed research journal, 2016, 9(2):146-152.

56. Nguyen D.H. Model-based testing of multiple GUI variants using GUI test generator / D.H. Nguyen, P.A. Strooper, S.J. Guy // ACM, 2010. [Electronic resource]. – Access mode: https://doi.org/10.1145/1808266.1808270.

57. 佚名.数说 十年风雨,电商从稚嫩走向成熟[J].中国商界, 2020(12):10.

58. Nagarajan M. A Multi-Period Model of Inventory Competition / M. Nagarajan, R. S. Rajagopalan // Social Science Electronic Publishing, - № 57(3), 2009. – Pp. 785-790. [Electronic resource]. – Access mode: https://doi.org/10.2139/ssrn.1140311.

59. Moise D. Luc Bougé.Improving the Hadoop map/reduce framework to support concurrent appends through the BlobSeer BLOB management system / D. Moise, G. Antoniu // ACM, 2010. [Electronic resource]. – Access mode: https://doi.org/10.1145/1851476.1851596.

60. Teranishi M. Computer system and storage system / M.Teranishi, H. Shibuya, S.Murayama et al // United States Patent: Pub. No.: US 2013/0212335 A1, 2013. [Electronic resource]. – Access mode: https://patentimages.storage.googleapis.com/84/cc/02/78cb2e5306e7bc/US20130212335A1.pdf.

61. Fan P. PC: a distributed transaction concurrency control protocol of multi-microservice based on cloud computing platform / P. Fan, J. Liu, W. Yin et al // Journal of Cloud Computing, 2020, 9(1):22. [Electronic resource]. – Access mode: https://doi.org/10.1186/s13677-020-00183-w.

62. Liu Y. Research achievements on the new generation Internet architecture and protocols / Y. Liu, J.P. Wu, Z. Zhang et al., 2013(11):25. [Electronic resource]. – Access mode: https://doi.org/10.1007/s11432-013-5021-4.

63. Bai Y. Dynamic reverse proxy chain generation for networks in data centers / Y.Bai, G.Guo, Y.Wang et al // 会议论文, 2020, 000(9226588). [Electronic resource]. – Access mode: https://doi.org/10.23919/APNOMS50412.2020.9237006.

64. Min H. Research and application of web cache technology / H. Min, Z. Wei-Dong, L.I. Zhong-Li // Computer Engineering and Design, 2003.

65. Bakshi K. Microservices-based software architecture and approaches / K. Bakshi //2017 IEEE Aerospace Conference.IEEE, 2017. [Electronic resource]. – Access mode: https://doi.org/10.1109/AERO.2017.7943959.

66. Che Z. Swarm intelligence-based optimization of machine learning models for diverse industrial problems: a comparative study / Z. Che, C. Peng, J. Wang // IOP Publishing Ltd, 2024. [Electronic resource]. – Access mode: https://doi.org/10.1088/1742-6596/2852/1/012001.

67. Lee J. Distributed Transaction Management Using Optimization Of Local Transactions / J. Lee, M. Muehle, J. Noh // United States Patent: Pub. No.: US20120167098A1, 2025. [Electronic resource]. – Access mode: https://doi.org/US20120167098 A1.

68. Casalaina M.C. Delta caching / M.C. Casalaina // BUSTECH 2014: The Fourth International Conference on Business Intelligence and Technology. [Electronic resource]. – Access mode: https://www.google.com/patents/US7406514 [Jul, 2008].