Textile & Leather Review

ISSN 2623-6281| www.tlr-journal.com | 10.31881/TLR

Research on the Application of Web-Based Linux Server Monitoring Platform in the Textile Machinery Industry

Jiantao Cui, Qinglin Ma, Wenheng Yang, Jie Zhang, Zihan Yuan, Zuhui Shen

How to cite: Cui J, Ma Q, Yang W, Zhang J, Yuan Z, Shen Z. Research on the Application of Web-Based Linux Server Monitoring Platform in the Textile Machinery Industry. Textile & Leather Review. 2025; 8:1140-1162. https://doi.org/10.31881/TLR.2025.1140

How to link: https://doi.org/10.31881/TLR.2025.1140

Published: 29 December 2025

Research on the Application of Web-Based Linux Server Monitoring Platform in the Textile Machinery Industry

Jiantao CUI, Qinglin MA*, Wenheng YANG, Jie ZHANG, Zihan YUAN, Zuhui SHEN

College of Software Engineering, Zhengzhou University of Light Industry, Zhengzhou 450001, Henan, China

*18521384185@163.com

Article

https://doi.org/10.31881/TLR.2025.1140

Received 23 May 2025; Accepted 4 July 2025; Published 29 December 2025

ABSTRACT

To support the digital operation and intelligent management of modern textile manufacturing systems, this paper designs and implements a web-based Linux monitoring platform oriented toward textile production environments. In large-scale textile enterprises, production management, quality control, and equipment coordination rely heavily on distributed server infrastructures that support textile machinery clusters and manufacturing information systems. Inefficient monitoring and delayed fault response in such environments can directly affect production stability and product quality.

The proposed platform integrates textile-oriented monitoring requirements with Linux server management technologies. It employs HTML5 and JavaScript to construct an interactive monitoring interface and adopts the WebSocket protocol to enable event-driven, low-latency data transmission for textile production support systems. A dual back-end architecture based on Django and Spring5 is implemented to manage system resources, monitor service status, and support alarm mechanisms under multi-node textile production workloads.

The platform enables real-time observation of server performance indicators associated with textile manufacturing systems, including resource utilization, service availability, and abnormal event detection. Experimental evaluation under simulated textile production monitoring scenarios demonstrates that the proposed solution improves monitoring responsiveness, reduces alarm latency, and enhances operational efficiency compared with traditional polling-based approaches.

The results indicate that the proposed platform provides an effective monitoring and management solution for textile production information systems and offers practical support for the digital transformation of textile manufacturing enterprises.

KEYWORDS

textile manufacturing, textile production, Linux server, monitoring platform, WebSocket

INTRODUCTION

Against the backdrop of accelerating digital transformation in the global manufacturing sector, traditional labor-intensive industries such as textiles are undergoing structural changes driven by information technology and data-centric management paradigms. Concepts including industrial digitalization, digital industrialization, and Industry 4.0 emphasize the integration of information systems into manufacturing processes to improve operational efficiency, resource utilization, and management transparency [1–3]. Within this context, textile enterprises are increasingly required to enhance their information infrastructure in order to remain competitive under conditions of rising labor costs, diversified customer demand, and intensified market competition [4–6].

Digital transformation in textile manufacturing is commonly reflected in the deployment of automated production equipment, intelligent sensing technologies, and data-driven enterprise management systems. As the core infrastructure supporting these systems, Linux servers play a critical role in hosting production management platforms, quality monitoring systems, and enterprise information services. With the expansion of system scale and functional complexity, server workloads and interdependencies within textile enterprises continue to increase. Consequently, the stability, responsiveness, and manageability of Linux-based server environments have a direct impact on the reliability of digital production and management systems [7,8].

Traditional server monitoring approaches, which are often based on manual inspection or low-frequency polling mechanisms, face limitations when applied to multi-node and heterogeneous enterprise environments. Such methods may suffer from delayed status updates, limited system visibility, and inefficient fault response, particularly when monitoring large numbers of servers and services simultaneously [9,10]. In addition, many existing monitoring solutions primarily focus on basic performance indicators and provide limited interactive visualization or flexible management capabilities, which constrains their applicability in complex industrial IT scenarios that demand timely situational awareness and efficient operational decision-making.

To address these challenges, this study focuses on the design and implementation of a web-based Linux server monitoring platform tailored for industrial IT environments in textile enterprises. The proposed platform emphasizes event-driven data acquisition and visualization rather than hard real-time industrial control. By adopting the WebSocket protocol, which supports persistent, full-duplex communication between clients and servers, the platform enables timely delivery of monitoring data and system status

updates compared with traditional request-response or polling-based architectures [11,12]. This communication mechanism improves monitoring responsiveness while maintaining compatibility with web-based management interfaces.

On the back end, the platform integrates a dual-framework architecture based on Django and Spring5 to balance development flexibility, scalability, and data processing efficiency. Django is responsible for system management functions, user access control, and data aggregation, while Spring5 supports concurrent data processing and monitoring-related analysis tasks. This modular design supports extensibility and practical deployment in enterprise-level server environments without introducing unnecessary system complexity. Overall, the proposed platform aims to improve monitoring responsiveness, system visibility, and operational efficiency for Linux-based infrastructures in textile enterprises, thereby supporting the broader digital transformation of industrial information systems.

REQUIREMENT ANALYSIS

User Requirement Analysis

To meet the operational and management needs of textile enterprises, the monitoring platform should not only support general Linux server management functions but also adapt to the characteristics of industrial IT environments, where multiple servers, application systems, and device-related data sources coexist. In such environments, servers are often deployed in a distributed manner to support production management systems, quality inspection platforms, logistics coordination, and data analysis services, which increases the complexity of monitoring and operation.

Real-Time Status Monitoring and Visualization

The platform should support continuous collection of key operational indicators related to servers and associated application systems, such as CPU utilization, memory usage, disk I/O, network traffic, and service status. These indicators should be visualized dynamically through dashboards and trend charts, enabling operators to observe system behavior over time, compare data across multiple nodes, and identify abnormal conditions at an early stage. Compared with traditional low-frequency polling mechanisms, timely monitoring updates are essential for reducing fault detection latency and improving operational awareness.

Multi-Node Coordination and Unified Management

Textile enterprises typically deploy multiple servers to support different business functions. The monitoring

platform should provide unified management across distributed nodes, allowing centralized observation of system status, configuration consistency checking, and resource utilization analysis. Such coordination helps operation and maintenance personnel manage heterogeneous server environments efficiently and reduces the risk of configuration errors and resource imbalance.

Fault Warning and Maintenance Support

By integrating monitoring data from servers and related subsystems, the platform should support abnormal event detection and early warning mechanisms based on predefined thresholds or rules. Detected anomalies can be linked to maintenance workflows, assisting operation and maintenance personnel in locating potential issues, recording maintenance history, and scheduling follow-up actions. This approach supports proactive operation and maintenance and helps reduce unplanned downtime in enterprise systems.

Security and Compliance Management

To satisfy enterprise-level data security and compliance requirements, the platform should support encrypted data storage, role-based access control, and hierarchical permission management. Different user roles, such as operators, maintenance personnel, and administrators, should be clearly defined to ensure that system access and operations comply with organizational security policies and industry regulations.

Technical Analysis

Front-End Service

The front-end interface is developed using web technologies such as HTML5, CSS3, and JavaScript to provide an interactive and user-friendly monitoring interface [13]. JavaScript enables dynamic data updates and responsive user interactions, while HTML5 provides a structured layout for presenting monitoring information. When server performance data or system status changes, the front-end interface updates charts and indicators dynamically, allowing operators to observe system behavior and trends in a timely manner. The front-end interaction workflow is illustrated in Figure 1.

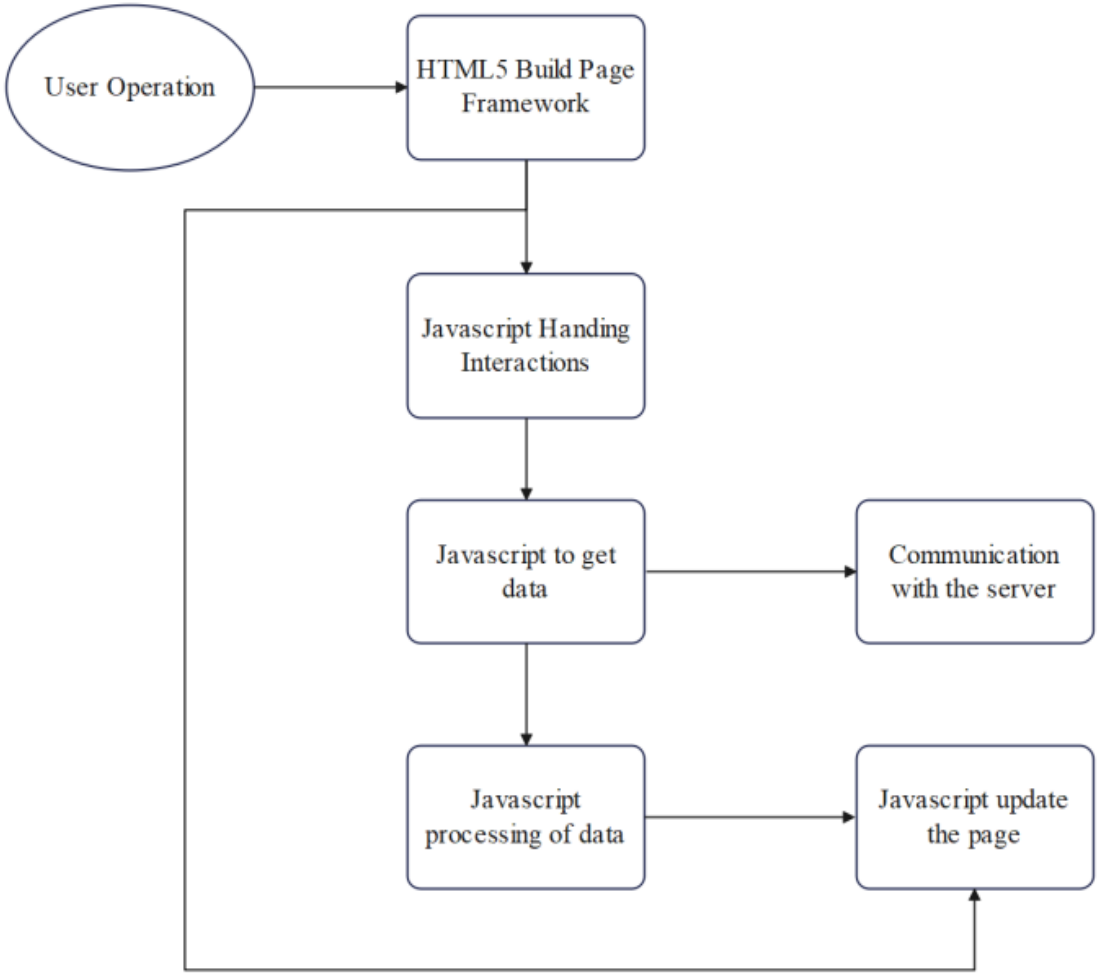

Figure 1. Front-end Interaction Operation Flowchart

To support timely and efficient data delivery, the platform employs the WebSocket protocol to establish persistent connections between the client and the server [14]. Unlike traditional polling-based approaches, WebSocket enables event-driven data transmission, allowing monitoring updates, system logs, and status changes to be pushed to the front end as they occur. This mechanism reduces unnecessary network overhead and improves responsiveness in multi-node monitoring scenarios. The WebSocket event handling process is shown in Figure 2.

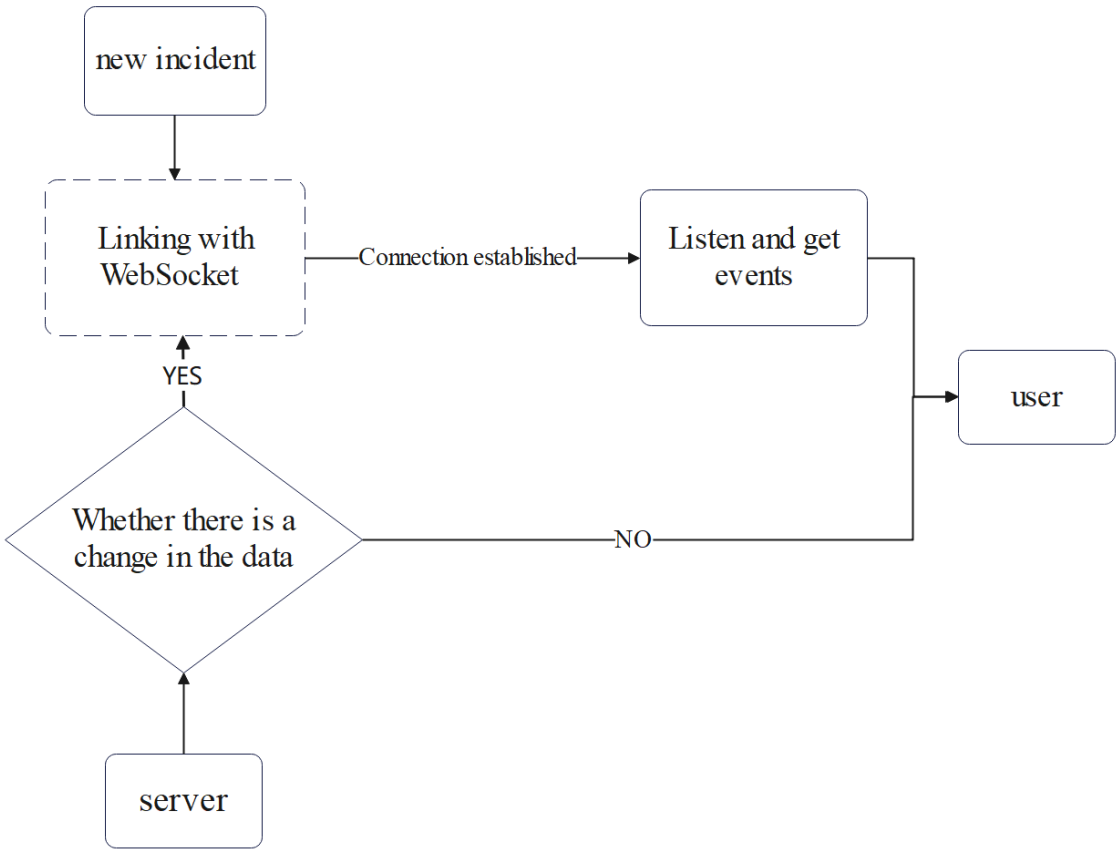

Figure 2. WebSocket Event Flowchart

Back-End Service

The back-end services are implemented using the Django framework, which provides stable and reliable support for system management functions such as data aggregation, configuration management, and user access control in enterprise environments [15]. Django serves as the primary coordination layer, handling request routing, permission management, and integration with data storage services.

To support concurrent data processing and analysis tasks, the platform integrates the Spring5 framework as a complementary back-end component. Spring5 is responsible for executing computationally intensive monitoring-related tasks, including statistical analysis of collected metrics and generation of system status evaluations and warning information. By separating coordination logic and data processing logic, the platform improves modularity and scalability.

The two frameworks interact through HTTP-based APIs using structured data formats, enabling flexible integration while maintaining clear functional boundaries. This dual-framework design supports system extensibility and facilitates future upgrades or replacement of individual components without affecting the overall architecture.

The platform utilizes a MySQL database to store monitoring data, configuration information, and user management records. Stored data include server performance indicators, system logs, permission settings, and monitoring thresholds. The database supports historical data analysis, visualization, and system configuration management, contributing to the reliability and maintainability of the monitoring platform [16]. The overall data service management structure is illustrated in Figure 3.

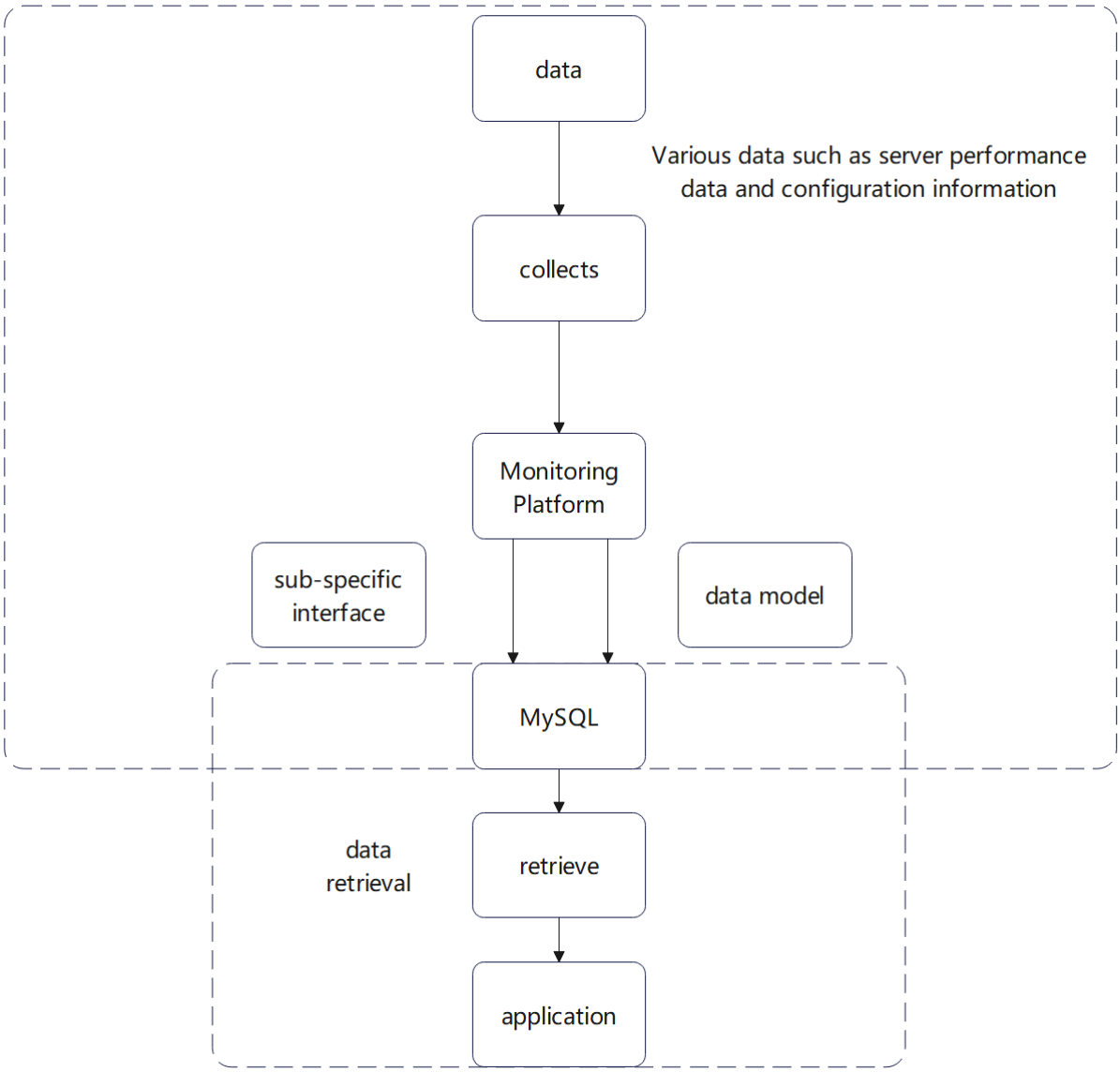

Figure 3. Data Service Management Diagram

Platform Design

Design of the System's Transmission Architecture

The proposed platform adopts a distributed monitoring architecture composed of multiple autonomous Linux monitoring systems (System1, System2, and System3). Each system independently collects and manages local operating system information and service status, while supporting centralized monitoring through a web-based interface. Communication between the web interface and each monitoring system is primarily achieved using the WebSocket protocol, which enables timely transmission of monitoring data and status updates. Secure Shell (SSH) is employed as a supplementary channel for remote system management, command execution, and secure data access, with strict authentication and permission control mechanisms in place [17].

The web interface establishes persistent WebSocket connections with the monitoring systems to receive real-time updates on system status, performance indicators, and service conditions. This bidirectional communication mechanism allows users to interact with the monitoring platform efficiently without relying on frequent polling requests. Data exchanged between the front end and the back end are serialized using structured formats, ensuring compatibility and extensibility of the platform.

SSH communication is utilized to support remote administration tasks such as information collection, configuration management, and log retrieval. The SSH-based interactions are restricted to authorized users and are designed to complement the WebSocket-based monitoring mechanism rather than to support real-time control functions [18]. This separation of monitoring and management responsibilities enhances system security and operational stability.

The platform continuously collects key kernel-level metrics, including CPU usage, memory utilization, network interface status, and file system information. In addition, the operational status of essential services, such as sshd, Docker, Kubernetes, and system logging services, is monitored to support early detection of abnormal conditions. The modular design of the platform facilitates scalability and allows additional monitoring nodes or functional modules to be integrated as system requirements evolve [19]. The overall system transmission architecture is illustrated in Figure 4.

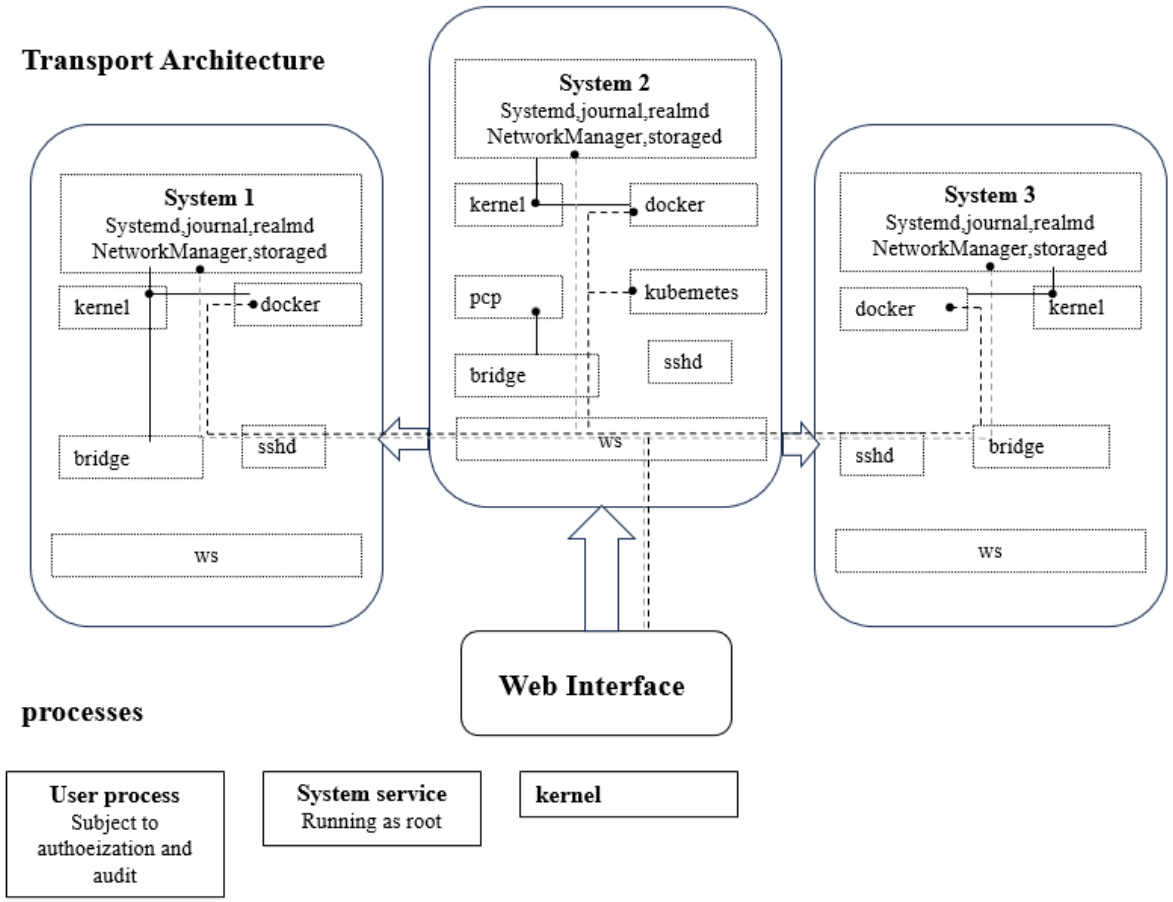

Figure 4. Overall system transmission architecture of the monitoring platform

Design of the Network and Its Interfaces

The platform provides two primary access methods for users: a web-based application interface and an SSH terminal interface. Through the web application, users connect to the monitoring platform via designated network ports, and interactive sessions are established using WebSocket-based communication mechanisms. This approach enables efficient delivery of monitoring information and responsive user interaction.

For advanced system management and maintenance operations, users can access the platform through an SSH terminal. In this mode, secure connections are established using standard SSH protocols, allowing authorized users to execute system commands and utilize command-line tools [20]. The application interface interacts with core system components and services through standardized system APIs, including

the kernel, system management services, logging facilities, container management components, and storage subsystems. The network and interface design is shown in Figure 5.

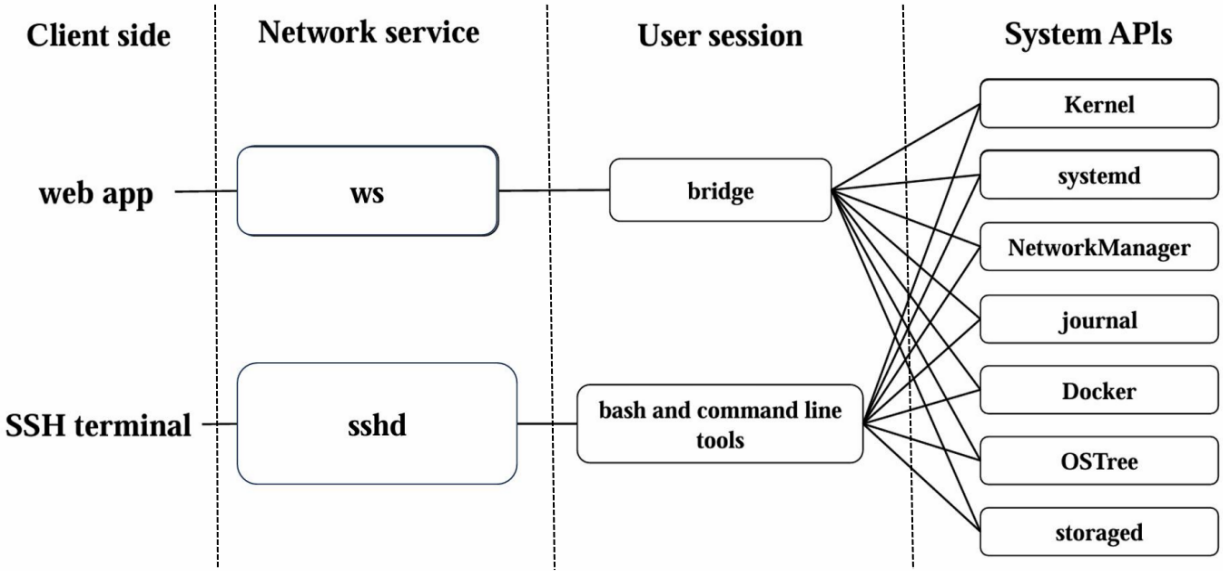

Figure 5. Network and interface design of the monitoring platform

Design of the Interface

Real-Time Performance Indicators

The platform provides real-time visualization of key system performance indicators, such as CPU utilization, memory usage, network traffic, and disk I/O. These indicators enable users to identify performance bottlenecks and abnormal system behavior promptly, thereby supporting efficient operation and maintenance activities [21]. The structure of the performance monitoring interface is shown in Figure 6.

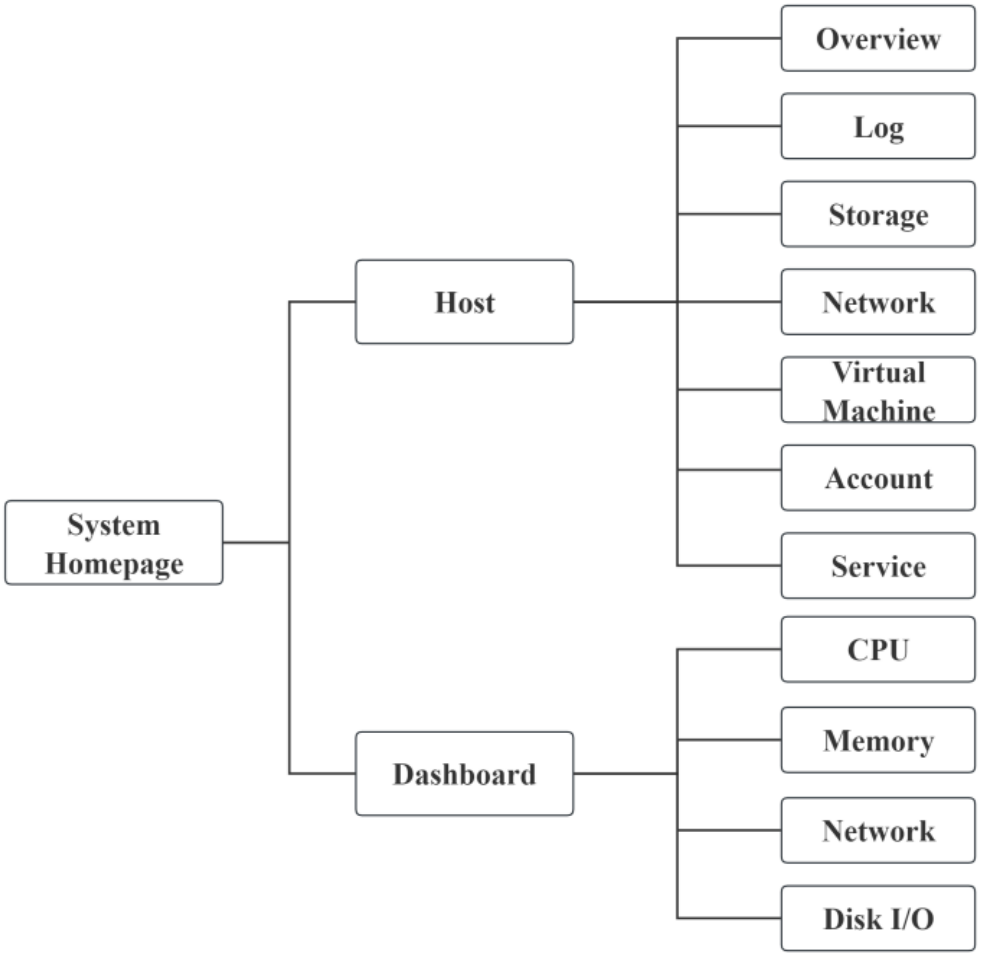

Figure 6. Example of the system performance monitoring interface

Visualization and Interactivity

To enhance usability and operational efficiency, the monitoring interface adopts a clear and modular design that emphasizes key information while avoiding unnecessary complexity [22]. Dynamic data updates allow users to observe system status changes continuously, while interactive visualization components—such as line charts, bar charts, and summary panels—support data exploration and comparative analysis.

Additional interactive features, including filtering, searching, customizable views, and configurable alert notifications, enable flexible management of monitoring data. Users can personalize interface layouts and monitoring perspectives to suit different operational scenarios, facilitating efficient decision-making and system supervision across diverse application contexts.

FUNCTIONAL DESIGN

System Monitoring

Key metrics, including CPU utilization, memory usage, disk I/O, and network traffic, can be monitored continuously to support timely diagnosis of performance bottlenecks and abnormal behavior. For example, the load of each CPU core can be displayed to assist in locating CPU-intensive services.

User Management

The platform supports user account management, including creating, deleting, and updating user permissions. It also enables the assignment of specific file and directory access rights according to operational requirements and security policies.

Network Configuration

The platform provides a centralized view of network interfaces on the server, including Ethernet and wireless interfaces. For each interface, information such as interface name, MAC address, connection status, and assigned IP address can be viewed and managed [23]. In addition, virtual network interfaces can be created and configured when needed (e.g., for traffic isolation or service-level separation).

Storage Management

Users can visualize disk partition layouts, including partition size, file system type, and mount point information. The platform supports basic storage operations such as creating partitions (e.g., after adding new disks) and resizing partitions where applicable, while maintaining data safety constraints. Storage usage can be monitored continuously, including used space, free space, and utilization rate, enabling proactive capacity planning and avoiding service disruptions caused by insufficient storage [24].

Automated Operation and Maintenance

The platform supports rule-based automation of routine maintenance tasks, such as periodic update checks, scheduled backup execution, and automated service start/stop policies. Automation reduces repetitive manual work and improves operational efficiency.

Service Control

The platform provides service status monitoring and configuration management, as well as basic lifecycle operations such as startup/shutdown control, resource observation, and log viewing. The service management workflow is illustrated in Figure 7.

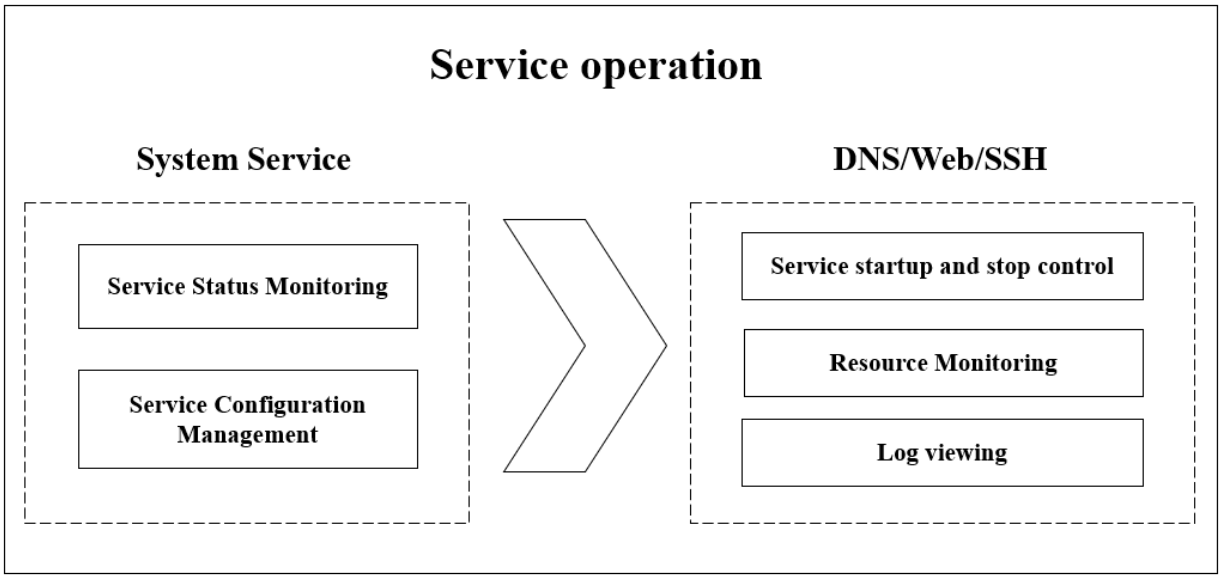

Figure 7. Functional structure of service management within the monitoring platform

TESTING OF SYSTEM

To evaluate the platform under scalable and repeatable conditions, a simulation-based test environment was constructed to emulate an enterprise-level multi-node server monitoring scenario. A cluster of 40 simulated nodes was deployed and organized into four groups of 10 nodes to represent heterogeneous server roles and workload patterns (e.g., database-like load, web-service load, batch-processing load, and mixed load). This configuration reflects typical multi-server deployments in industrial IT environments and enables controlled stress testing.

During stress testing, each node generated monitoring events (e.g., threshold crossings of CPU, memory, disk, or network indicators) and transmitted monitoring records to the platform at a sustained rate. Unless otherwise specified, each monitoring record used a fixed payload size (512 bytes) to ensure consistent measurement. All experiments were repeated three times under the same configuration to verify stability and repeatability of results.

Functional Testing

Functional testing was conducted to verify that core modules—system monitoring, user management, network configuration, storage management, and service control—operate correctly under both normal and stressed conditions. Stress scenarios included high CPU utilization, memory pressure, and intensified I/O to emulate typical overload states in enterprise server environments.

For alarm responsiveness evaluation, an “abnormal event” was defined as a monitored metric crossing a predefined threshold. The alarm delay was measured as the elapsed time from the moment the threshold was detected at the back end to the time the corresponding alarm was rendered on the web interface. Results indicate that the proposed platform maintained a stable alarm delay of approximately 300 ms, representing a substantial reduction compared with a traditional polling-based baseline (average approximately 1.8 s). These measurements were obtained using platform logs and timestamped event records. The fault warning delay comparison is shown in Figure 8.

In addition, the user management module was tested for permission update correctness and auditability. Permission modifications were validated using database transaction logs and audit trails to ensure consistency and traceability.

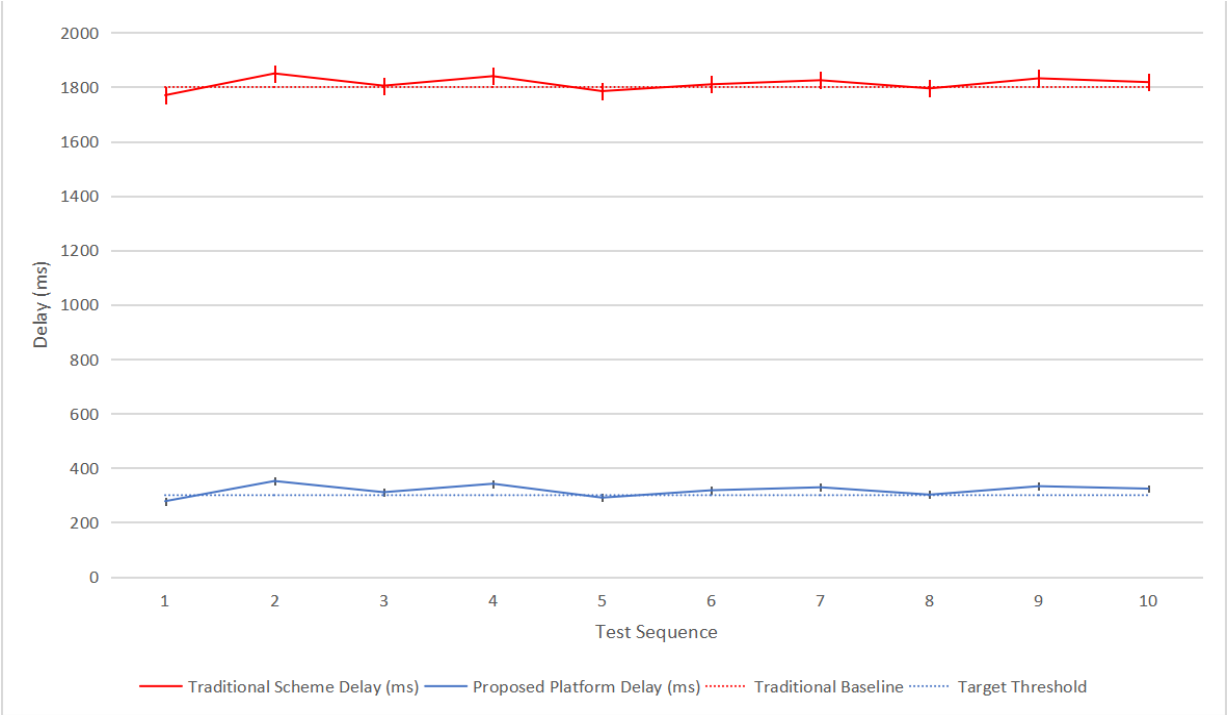

Figure 8. Evaluation of abnormal event alarm latency under simulated load conditions

Performance Testing

Performance testing focused on verifying platform stability and resource consumption under different workload and network conditions [25]. Tests were conducted on a server environment running Ubuntu 20.04 with an Intel i5-7200 (2.50 GHz) CPU and 16 GB RAM. Monitoring dashboards were accessed continuously while the platform processed incoming monitoring data streams and alarm events.

Resource utilization measurements (CPU, memory, disk I/O, and network traffic) were collected during

sustained operation and under stress loads. The results demonstrate that the platform remained stable without abnormal resource spikes under the tested configurations. Representative monitoring charts are shown in Figure 9 and Figure 10.

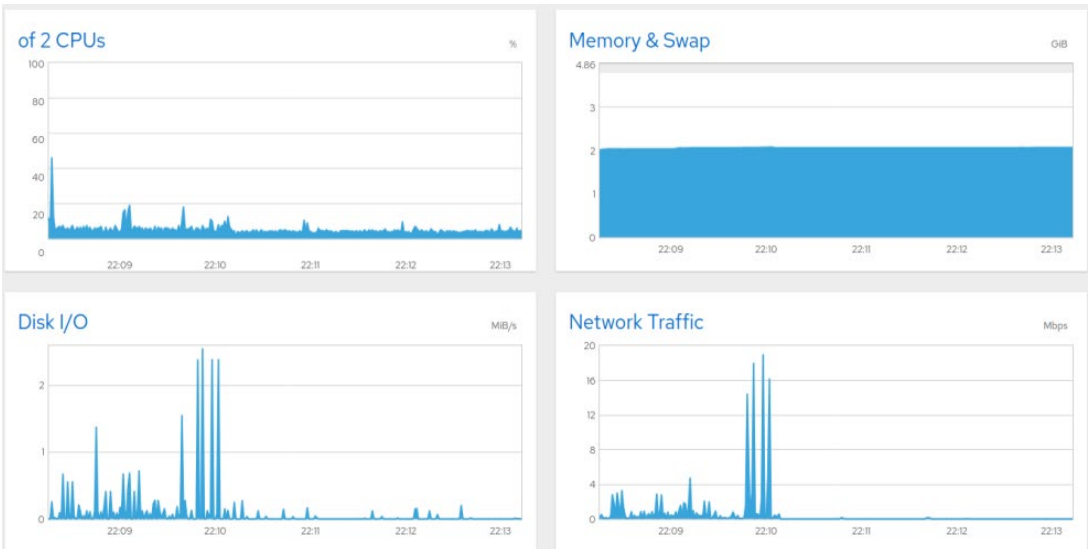

Figure 9. System performance indicators measured under network-connected conditions

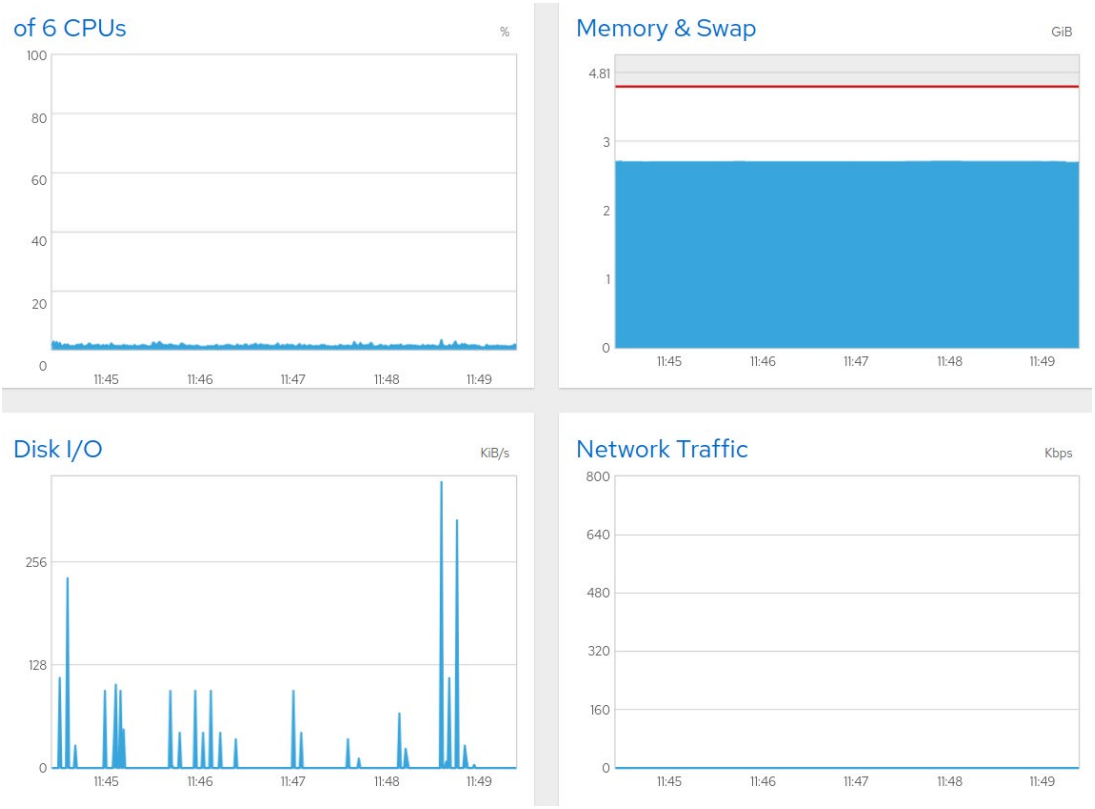

Figure 10. System performance indicators measured under isolated (no-network) conditions

Response Time

To evaluate user-perceived responsiveness of the monitoring dashboard, response time was measured from the moment a monitoring update was generated by the back end to the moment the updated values and charts were fully rendered on the web interface. Experiments were performed under a controlled network environment (latency < 1 ms, bandwidth 1 Gbps) using the same hardware configuration described above. JMeter was used to simulate concurrent dashboard requests, and for each load level, 100 consecutive requests were sampled.

The proposed platform consistently achieved lower dashboard response time compared with the polling-based baseline across repeated trials. The comparison results are shown in Figure 11. Summary statistics of response time are shown in Figure 12, where the average response time of the polling-based baseline was approximately 137 ms, while the proposed platform achieved approximately 50 ms under the same test setting.

A statistical significance test was conducted on the collected samples to verify that the observed difference was not due to random variation. Results indicate that the reduction in response time is statistically significant ($p < 0.001$). These findings demonstrate that the WebSocket-based event-driven update mechanism improves dashboard responsiveness under concurrent access.

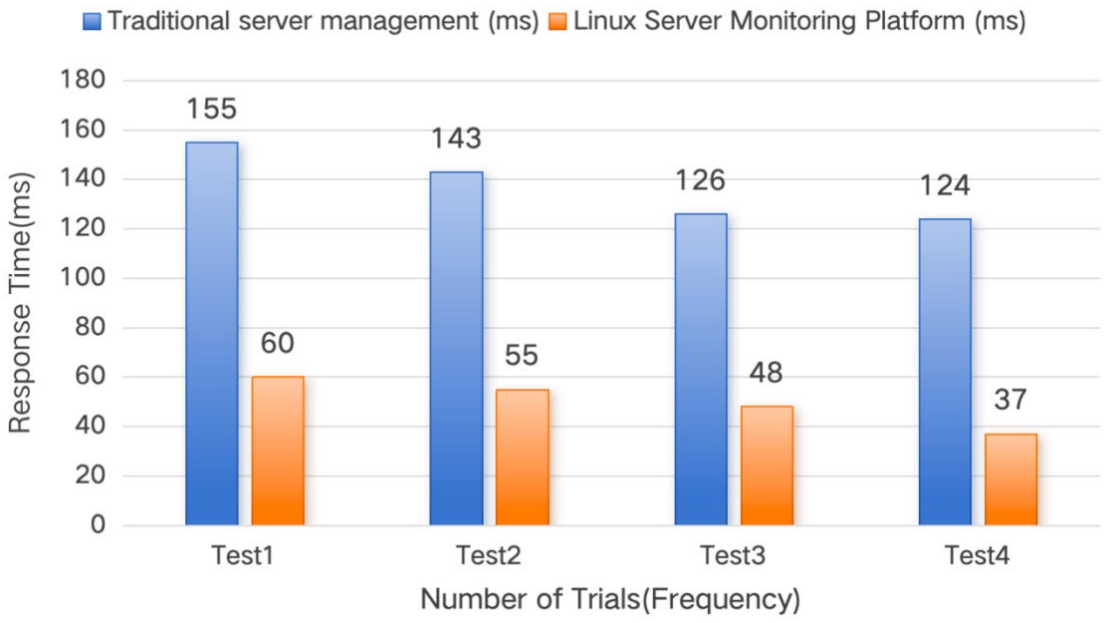

Figure 11. Comparison of system response time under different monitoring architectures

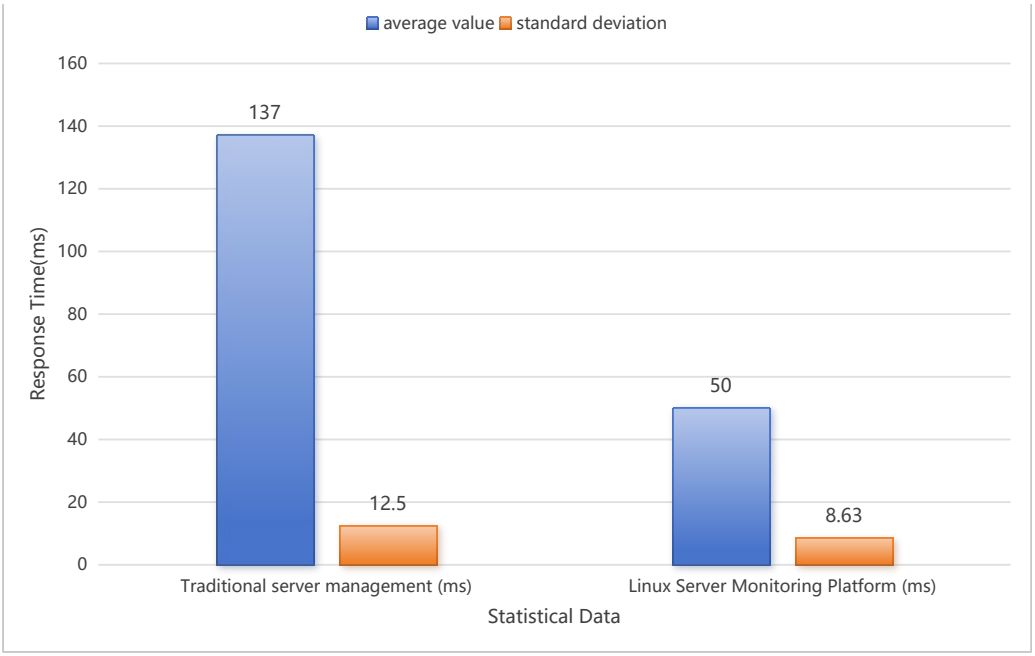

Figure 12. Statistical analysis of response time measurements

Django-Spring5 Cross-Framework Communication Latency Testing

To evaluate the performance of the dual-framework architecture, cross-framework communication latency between Django and Spring5 was tested using an HTTP API (JSON). A total of 10,000 requests were generated via JMeter, and the end-to-end latency from request submission to response reception was recorded. Nginx was configured as a reverse proxy to improve connection stability and support concurrent request handling.

The experiment observed an average cross-framework response latency of approximately 7.5 ms under the tested configuration, indicating that the modular architecture introduces limited overhead and remains suitable for real-time monitoring data processing tasks in enterprise environments.

Textile Machinery Vibration Data Analysis Throughput Testing

To evaluate the effectiveness of data preprocessing and transmission optimization, comparative experiments were conducted under the same simulated multi-node monitoring workload. A data aggregation and compression strategy was applied to reduce redundant transmissions and to improve the efficiency of front-end data rendering.

Experimental results show that network bandwidth consumption was reduced from approximately 15.2 MB/s in the unoptimized configuration to 0.68 MB/s after optimization. In terms of visualization performance, the dashboard rendering frame rate increased from 8 fps to 48 fps, indicating more stable

and responsive front-end updates. In addition, the algorithmic anomaly detection latency, defined as the time between aggregated data arrival and abnormal event labeling by the back-end analysis module, decreased from 320 ms to 105 ms, reflecting improved processing efficiency. The comparative results are summarized in Table 1.

Table 1. Performance Optimization Comparative Results

Performance Indicator	Baseline (Without Aggregation)	Optimized (With Edge Aggregation)	Improvement
Network bandwidth usage	15.2 MB/s	0.68 MB/s	95.5% reduction
Aggregated data refresh latency (event-to-UI)	> 1,000 ms	< 200 ms	Significant reduction
Browser UI frame rate (Chrome)	8 fps	48 fps	Stabilized near display refresh rate
Algorithmic anomaly detection latency	320 ms	105 ms	67.2% reduction

Note: All measurements were obtained under identical experimental conditions. Network bandwidth values represent average sustained throughput during continuous monitoring.

Future work will explore extending the monitoring platform to integrate with additional industrial data sources and edge computing capabilities. Potential directions include lightweight sensor integration for collecting device-related operational signals and deploying edge-side preprocessing for bandwidth reduction and faster anomaly detection in complex workshop environments.

In addition, multi-protocol gateways may be investigated to improve interoperability with heterogeneous equipment and legacy systems. Advanced analytics methods (e.g., signal processing and filtering techniques) could be integrated to enhance anomaly detection accuracy and support predictive maintenance applications. These directions represent potential extensions and are not part of the validated implementation presented in this paper. The technology roadmap is shown in Figure 13.

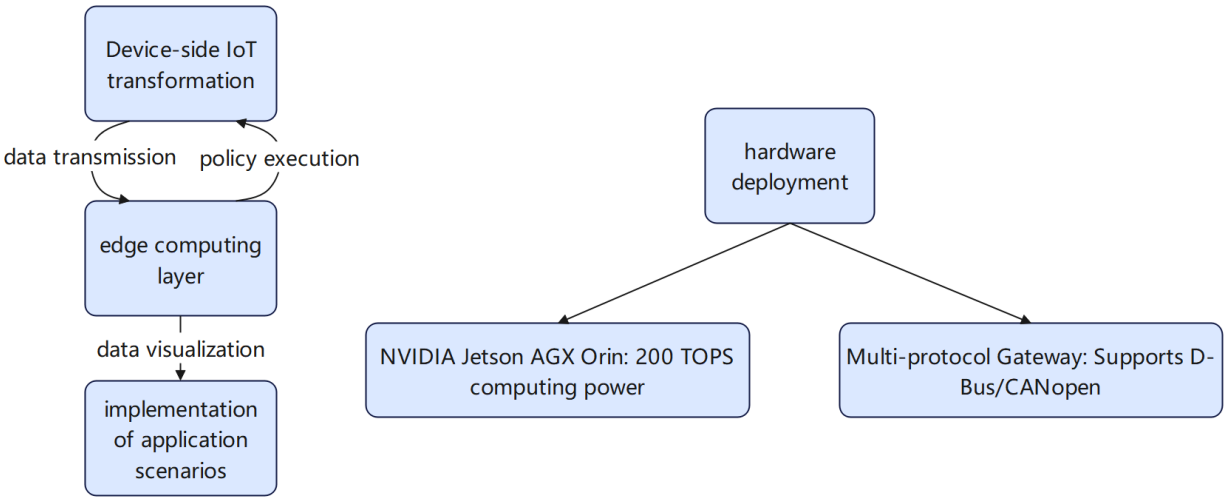

Figure 13. Technology Roadmap for IoT and Edge Computing Integration

Analysis of Economic Benefits

To provide a preliminary view of potential economic value, an estimated cost-benefit analysis was conducted based on a pilot-like operational scenario. The platform may reduce routine operation and maintenance workload through centralized monitoring, automation functions, and faster alarm response, which can translate into lower labor costs and reduced downtime risk. Improvements in resource visibility and utilization may also support more efficient capacity planning, potentially reducing unnecessary hardware expansion.

The economic figures shown in Figure 14 are provided as illustrative estimates for reference and should not be interpreted as statistically validated results across a broad population of enterprises. Future work will incorporate longer-term deployment data and clearer baselines to strengthen the economic evaluation.

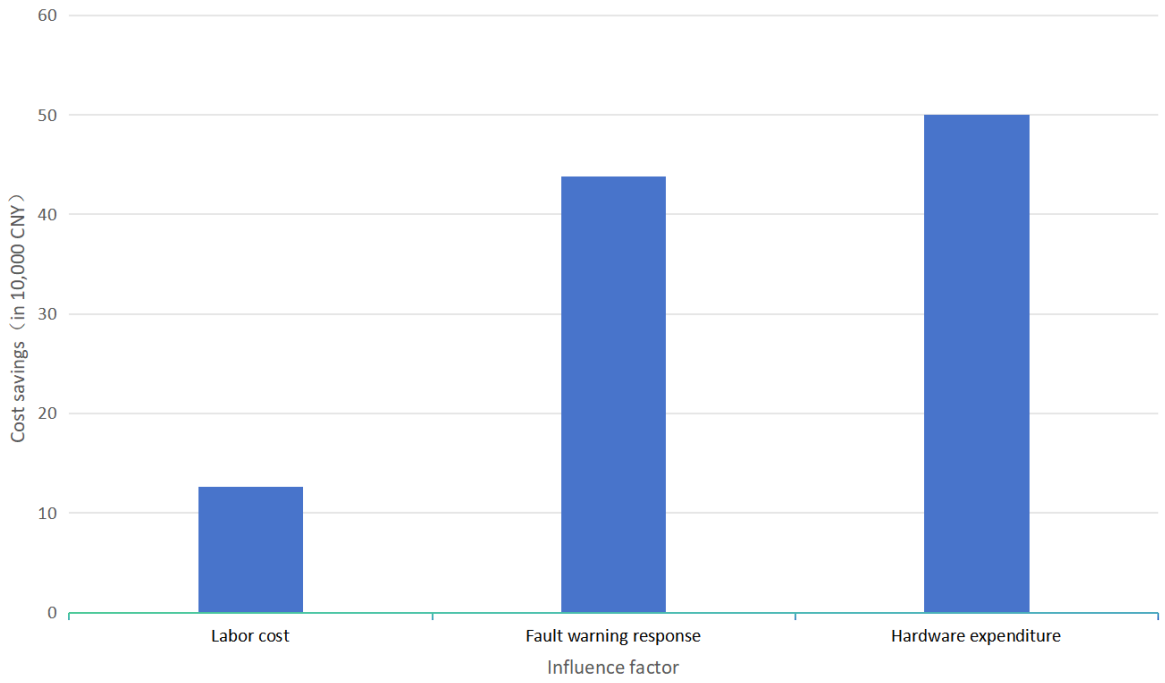

Figure 14. Analysis of economic benefits after platform deployment

CONCLUDING REMARKS

This paper presents a web-based Linux server monitoring platform tailored for industrial IT environments in textile enterprises, addressing challenges such as low monitoring responsiveness and the operational complexity of traditional server management approaches. The platform adopts HTML5/JavaScript and WebSocket on the front end to enable event-driven updates and interactive visualization, while the back end integrates Django and Spring5 to support scalable data processing, user management, and modular system services.

Experimental results demonstrate that the proposed platform improves dashboard response time and alarm responsiveness compared with polling-based baselines, while maintaining stable resource utilization under stress conditions. The platform also provides practical functions for system monitoring, access control, network and storage management, automated operation and maintenance, and service control, indicating strong applicability for enterprise-level operation and maintenance workflows.

Overall, the proposed design offers an extensible solution for digital operation and maintenance of Linux-based infrastructure. The architecture and implementation experience may be adapted to other industrial and enterprise IT monitoring scenarios with similar requirements.

In future work, the platform can be further extended in two key directions. First, a modular and microservice-oriented architecture can be adopted to improve maintainability, scalability, and independent

deployment of functional components. Second, IoT and edge computing technologies can be explored to integrate additional data sources from industrial environments, where edge nodes may perform lightweight preprocessing (e.g., filtering, aggregation, and compression) to reduce bandwidth consumption and improve monitoring responsiveness.

In addition, the monitoring network can be expanded to support broader production-site visibility, and data-driven methods can be introduced to enhance operational intelligence. For example, machine learning techniques may be used for anomaly detection, capacity forecasting, and early warning of infrastructure or service failures, thereby supporting proactive operation and maintenance. These extensions represent potential research and engineering directions and will be validated in future deployments to further support the digital transformation of textile enterprises and other manufacturing industries.

Author Contributions

Conceptualization – Jiantao CUI and Qinglin MA; methodology – Qinglin MA; formal analysis – Wenheng YANG and Jie ZHANG; investigation – Zihan YUAN; resources – Qinglin MA; writing-original draft preparation – Jiantao CUI, Wenheng YANG and Zihan YUAN; writing-review and editing – Jie ZHANG and Zuhui SHEN; visualization – Zuhui SHEN. All authors have read and agreed to the published version of the manuscript.

Conflicts of Interest

The authors declare no conflict of interest.

Funding

The authors acknowledge the Specialized and Creative Integration Characteristic Demonstration Courses (Second Batch) in Henan Province (Comprehensive Practice of Computer Network Technology, Grant:74), the Key R&D and promotion projects in Henan Province (Grant: 252102211105) and the Innovation Training Program for College Students in Henan Province (Grant: 202410462058).

Acknowledgements

The authors would like to thank associate prof. Jiantao Cui and Wei Huang, College of Software Engineering, Zhengzhou University of Light Industry for providing insight and expertise that greatly assisted the research. The authors would like to gratefully acknowledge and thank associate professor Jiantao Cui and all members of the project team for helping to collect the software and hardware data.

REFERENCES

[1] Veile JW, Kiel D, Müller JM, Voigt KI. Lessons learned from Industry 4.0 implementation in the German manufacturing industry. Journal of Manufacturing Technology Management. 2020; 31(5):977-997.

[2] Xu LD, Xu EL, Li L. Industry 4.0: State of the art and future trends. International Journal of Production Research. 2018; 56(8):2941-2962.

[3] Frank AG, Dalenogare LS, Ayala NF. Industry 4.0 technologies: Implementation patterns in manufacturing companies. International Journal of Production Economics. 2019; 210:15-26.

[4] Manglani H, Hodge GL, Oxenham W. Application of the internet of things in the textile industry. Textile Progress. 2019; 51(3):225-297.

[5] Kim M, Ahn J, Kang J, Kim S. A systematic review on smart manufacturing in the garment industry. Fashion and Textile Research Journal. 2020; 22(5):660-675.

[6] Haq UN, Khan MMR, Khan AM, Hasanuzzaman M, Hossain MR. Global initiatives for industry 4.0 implementation and progress within the textile and apparel manufacturing sector: A comprehensive review. International Journal of Computer Integrated Manufacturing. 2025; 38(12):1637-1662.

[7] Turnbull J. The Art of Monitoring. James Turnbull. 2014. Available from: https://books.google.com.hk/books?hl=zh-TW&lr=&id=w5QfDAAAQBAJ&oi=fnd&pg=PA1&dq=%5B1%5D%09Turnbull+J.+The+art+of+monitoring.+James+Turnbull.+2014.&ots=iHEfEpI40j&sig=MyI-1jvEc1kmgna4DLIyKCBNGW8&redir_esc=y#v=onepage&q&f=false

[8] Boras M, Balen J, Vdovjak K. Performance evaluation of linux operating systems. Proceedings of the 2020 International Conference on Smart Systems and Technologies (SST); 14-16 October 2020; Osijek, Croatia. New York, NY, USA: IEEE; 2020. p. 115-120.

[9] Repetto M. Adaptive monitoring, detection, and response for agile digital service chains. Computers & Security. 2023; 132:103343.

[10] Cherrared S, Imadali S, Fabre E, Gössler G, Yahia IGB. A survey of fault management in network virtualization environments: Challenges and solutions. IEEE Transactions on Network and Service Management. 2019; 16(4):1537-1551.

[11] Pautasso C, Zimmermann O, Leymann F. Restful web services vs." big" web services: making the right architectural decision. Proceedings of the 17th International Conference on World Wide Web; 21-25 April 2008; Beijing, China. 2008. p. 805-814.

[12] Fette I, Melnikov A. The WebSocket Protocol. 2011. doi: 10.17487/rfc6455. Available from: https://www.rfc-editor.org/rfc/rfc6455.html

[13] Kuo LW, Chang TY, Lai CC. Affective Psychology and Color Display of Interactive Website Design. Displays. 2022; 71:102134.

[14] Mitrovic N, Dordevic M, Veljkovic S, Dankovic D. Implementation and Testing of Websocket Protocol in ESP32 Based IOT System. Facta Universitatis-Series Electronics and Energetics. 2023; 36(2):267-284.

[15] Carvallo A, Jorquera I, Aspillaga C. CoTranslate: A Web-Based Tool for Crowdsourcing High-Quality Sentence Pair Corpora. SoftwareX. 2023; 23:101508.

[16] Petrov P, Kuyumdzhiev I, Dimitrov G, Kremenska A. Relative Performance of Various Types of Repositories for MySQL Archive Backup and Restore Operations. International Journal of Online and Biomedical Engineering. 2022; 18(13):152-159.

[17] Toledo S. SSH Tunneling to Connect to Remote Computers. Software Impacts. 2023; 17:100545.

[18] Zhong Y, Chen PF, Zhang HX. ESX: A Self-Generated Control Policy for Remote Access with SSH based on eBPF. IEEE Access. 2025; 13:6487-6506.

[19] Kirilov N. Comparison of WebSocket and Hypertext Transfer Protocol for Transfer of Electronic Health Records. Studies in Health Technology and Informatics. 2024; 313:124-128.

[20] Sentanoe S, Reiser HP. SSHkex: Leveraging Virtual Machine Introspection for Extracting SSH Keys and Decrypting SSH Network Traffic. Forensic Science International-Digital Investigation. 2022; 40:301337.

[21] Xing YK, Shell J, Fahy C, Xie TD, Kwan HY, Xie WQ. Web XR User Interface Research: Design 3D Layout Framework in Static Websites. Applied Sciences-Basel. 2022; 12(11):5600.

[22] Lei S. Modeling an Web Community Discovery Method With Web Page Attraction. Journal of Intelligent & Fuzzy Systems. 2021; 40(6):11159-11169.

[23] Azadiabad S, Khendek F, Toeroe M. Runtime Adaptation Framework for Fulfilling Availability and Continuity Requirements of Network Services. IEEE Transactions on Network and Service Management. 2023; 20(3):2259-2282.

[24] Nguetchouang K, Bitchebe S, Dubuc T, Callau-Zori M, Hubert C, Olivier P, Tchana A. SVD: A Scalable Virtual Machine Disk Format. IEEE Transactions on Cloud Computing. 2024; 12(2):684-696.

[25] Zolfaghari R. Energy-Performance Aware Virtual Machines Migration in Cloud Network by Using Prediction and Fuzzy Approaches. Engineering Applications of Artificial Intelligence. 2024; 131:107825.