Scholar Publishing Group
International Journal of Multimedia Computing
https://doi.org/10.38007/IJMC.2026.070204
ISSN 2789-7168 Vol. 7, Issue 2:28-37

Research on Load Balancing Strategy for Cloud Platforms Based on Docker Containers

Wenhao Song

Information Networking Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 15213, United States

Keywords: Docker containers; cloud platform; load balancing; resource prediction; multi-objective scheduling; quality of service

Abstract: Given the problems of resource skew, tail latency amplification and frequent migration under bursty traffic in heterogeneous nodes and cross-node network latency of Docker container cloud platforms, a predictive multi-metric load balancing strategy has been designed. This strategy jointly uses CPU, memory, network, queues and energy consumption as the state, applies exponential smoothing for short-term request intensity prediction, and conducts container placement and request distribution via feasible domain filtering, dynamic weight scoring, hysteresis migration, and abnormal node deweighting. Based on public research papers and experimental data from the last three years, this study also has some secondary analysis. According to the above results, at fewer than 16 concurrent requests, a 30ms inter-node latency can reduce throughput by 58.0% under a uniform Pod distribution; localisation and resource-aware scheduling can mitigate this drop significantly. The proposed method takes into account the constraints of load dispersion, P95 latency and migration cost simultaneously to offer an interpretable and deployable load balancing framework for Docker Swarm and other compatible orchestration platforms.

1 Research Background and Objectives of Load Tilt in Docker Container Cloud Platforms

Containers reduce the overhead of virtualisation by sharing the host kernel and can run microservices on cloud platforms with small images, fast startup times, and high deployment density. However, a lightweight Design does not automatically eliminate resource contention: if a large number of requests are directed to a small number of containers over a short period, the CPU and memory capacities of these nodes may become unbalanced, or when service calls cross high-latency links, a round-robin strategy might treat "request quantity balancing" as "system load balancing", resulting in queuing at hot nodes, increased tail latency, and SLA violations.

Recently, most research has moved away from a single-CPU threshold to multi-resource cooperation. Qian and others proposed a differentiated resource weighting mechanism for the OpenStack and Docker converged environment, and showed that the matching of container type with node resources directly affects the load dispersion inside and outside the node [1]. A review of Kubernetes scheduling algorithms further shows that although general rules, multi-objective

optimisation, artificial intelligence scheduling and autoscaling have addressed some problems, the trade-off between dynamic load, migration overhead and interpretability has not been fully resolved simultaneously [2]. Therefore, this paper puts forward a predictive multi-index load balancing strategy for Docker container cloud platforms based on a closed-loop system of "prediction-scoring-execution-feedback". Reduce load variance, P95 response latency, abnormal forwarding ratio and invalid migration number without changing the business image.

The primary contributions of this paper are as follows: establishing a unified index for multi-resource status and imbalance; enhancing the foresight of sudden business surges through short-term load forecasting; designing an all-encompassing scoring function with constraints on network locality, node health and migration hysteresis; and analysing the quantitative effect of cross-node latency and scheduling strategy based on publicly available real experimental data. In order to systematically present the current situation of research, relevant problems and strategy models for deployment in this paper, this paper will be structured as follows: 1.

2 Advances in Multi-Resource Load Balancing Technology for Docker Container Cloud Platforms

2.1 Rule-based Scheduling Method Based on Joint CPU-Memory-Network Awareness

Traditional Round Robin, Random, and Least Connection are simple to implement and have low decision overhead, so they are suitable for environments with homogeneous nodes and requests of similar service time. They do not know about container resource profiling, cache locality and network topology. Wang and others modelled edge Kubernetes scheduling as a reinforcement learning problem and enhanced load balancing for peak task deployment by using a reward function that minimised the response time [3]. The above methods are adaptive, but training stability, exploration risk, and online inference costs restrict their direct application in the production control plane.

Network Latency is also relatively high in a cross-node environment. Kim and Kim's KubeEdge test found that uniform forwarding does not always result in higher throughput; at a latency of 15-30ms between edge nodes, local processing can be employed to reduce remote forwarding and stabilise the response time [4]. Therefore, the scheduler needs to determine both the location of the container and the destination server for the request; otherwise, optimisation of only the placement or only server load may lead to a local optimum.

2.2 Multi-objective Programmable Scheduling under Delay-Energy Consumption-Migration Constraints

Alelyani and others jointly modelled microservice replica placement and network traffic and resource utilisation, and used an improved particle swarm optimisation algorithm for scheduling. They verified the effectiveness of multi-objective placement in simulations on the Alibaba and Google datasets [5]. This study highlighted the problems of service dependency and network traffic, but its optimisation process is better suited for a long time scale and cannot be continuously applied to bursty traffic at the second level.

Load balancing for containers should also consider energy consumption. Algarni et al. performed AR, ARIMA and ETS predictions on the power consumption sequence of Docker containers to show that fine-grained power consumption can be predicted in advance using time series models [6]. It can be used to add energy consumption to the score function, but the prediction of energy consumption does not solve the problem of request distribution among nodes. Qiao and others proposed EdgeOptimizer, which can perform online verification and switching of Kubernetes

scheduling algorithms in a modular way to improve the repeatability of experiments and the programmability of algorithms [7]. It has a good engineering platform, but the specific multi-index decision rules are not yet available.

Rabieyan and others used integer programming and container migration algorithms to increase resource utilisation and reduce server energy consumption, and based on statistical analysis, verified that the multi-objective model outperformed the benchmark method [8]. The above studies have found that migration can correct long-term resource imbalances; however, without a hysteresis threshold, too frequent migration will overload the network and damage the cache, resulting in increased short-term jitter. Gómez-González and others proposed the customised Kubernetes scheduler Chronos to reduce network latency, disk write operations and CPU-intensive task response time on a real single-board computer testbed [9]. Zhang and others have combined real-time health indicators, Docker and Spring Cloud fault recovery to show that load balancing and high availability mechanisms need to be designed in coordination [10].

Table 1: Comparison of Static Rule-Based, Heuristic Optimization and Reinforcement Learning Load Balancing Methods

Method family	Primary signal	Online adaptability	Network awareness	Main strength	Typical limitation
Round Robin	Request count	Low	No	Minimal overhead	Ignores heterogeneity
Least Connection	Active sessions	Medium	Limited	Fast response to queue growth	Service-time bias
Resource-weighted	CPU / memory / I/O	Medium	Optional	Balances heterogeneous nodes	Reactive only
Metaheuristic placement	Multi-objective cost	Low–Medium	Yes	Near-global placement	High solve time
Reinforcement learning	State-action reward	High	Configurable	Adaptive policy	Training and safety costs
Proposed predictive strategy	Forecast + resources + topology	High	Yes	Explainable closed loop	Needs telemetry quality

Table 1 shows that static algorithms have the advantage of low overhead, but their observed variables are limited; heuristic and reinforcement learning methods can handle multiple objectives, but they have problems of solution latency and training security, respectively. This paper uses a "lightweight prediction + explicit scoring" method to make each weight interpretable and auditable, and introduces a hysteresis mechanism to suppress scheduling oscillations; thus, it meets the high-stability requirements of Docker cloud platforms for the control surface.

3 Problems with Sudden Traffic Surges and Docker Load Balancing Across Heterogeneous Nodes

3.1 Normalisation and Dynamic Weighting of Multidimensional Resource Indicators

CPU utilisation, memory usage, network bandwidth, disk I/O and activity queues are all different types that change over time. Only the CPU threshold can be used; otherwise, memory-intensive containers will be placed on nodes with idle CPUs but limited memory, and if fixed weights are set,

low-bandwidth nodes can become hotspots due to network bottlenecks. Therefore, all the indicators need to be normalised, and their respective weights should change dynamically based on the type of container and its running status.

$$U_i(t) = \alpha(t) \cdot \frac{C_i(t)}{C_i^{max}} + \beta(t) \cdot \frac{M_i(t)}{M_i^{max}} + \gamma(t) \cdot \frac{B_i(t)}{B_i^{max}} + \delta(t) \cdot \frac{Q_i(t)}{Q^{max}} \quad (1)$$

$U_i(t)$ is the total load of node i at time t in equation (1), and C_i, M_i, B_i and Q_i represent CPU, memory, network bandwidth and active queue respectively; the quantity with a superscript max is the node capacity or control limit; α, β, γ and δ are weights that vary according to the container profile, and the sum of the four is 1.

3.2 Masking of Node Load Tilting by Average Utilization

Although the cluster has a relatively low average CPU utilisation, it is not guaranteed to be operating normally; some nodes may be close to full capacity while others are idle. The standard deviation of the total load is used to reflect the disparity among nodes, and the scheduling goal is thus shifted from "increasing average utilisation" to "reducing node disparity under quality-of-service constraints".

$$J(t) = \sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} [U_i(t) - \bar{U}(t)]^2} \quad (2)$$

As shown in Equation (2), J(t) is the load dispersion, N is the number of working nodes, and $\bar{U}(t)$ is the average load. The smaller J(t) is, the more even the resource distribution. J(t) = 0 does not have to hold strictly; otherwise, forcing uniformity may introduce cross-node network latency or extra migration overhead.

3.3 Insufficient Response to Sudden Requests Due to Monitoring Sampling Lag

Periodically collect the monitoring index every 5-30 seconds. By the time the CPU exceeds the threshold for expansion or migration, a queue will have already formed. To reduce the lag of observation, an exponential moving-average prediction of the service arrival rate is used, and this predicted value is included in node scoring.

$$\hat{\lambda}_s(t+1) = \eta \cdot \lambda_s(t) + (1-\eta) \cdot \hat{\lambda}_s(t) \quad (3)$$

As shown in equation (3), $\lambda_s(t)$ is the observed request arrival rate of service s in the current window, $\hat{\lambda}_s(t+1)$ is the predicted value for the next window, and η is a smoothing coefficient. The more sudden the service, the higher the weight given to new observations should be; the more stable the service, the less noise suppression by η is required.

Figure 1 analysis: At 16 concurrent requests, the throughput of all three Pod distributions at 0 ms is around 1.68-1.80 kreq/s. When the inter-node latency reaches 30 ms, the throughput of the 4-4-4 uniform distribution is only 755 req/s; at this time, the 10-1-1 scenario has reduced by only 2.5%. Based on the above results, forwarding at the per-Pod level increases the cost of remote communication; therefore, load balancing in cloud platforms with heterogeneous networks should consider topological distance and request source locality.

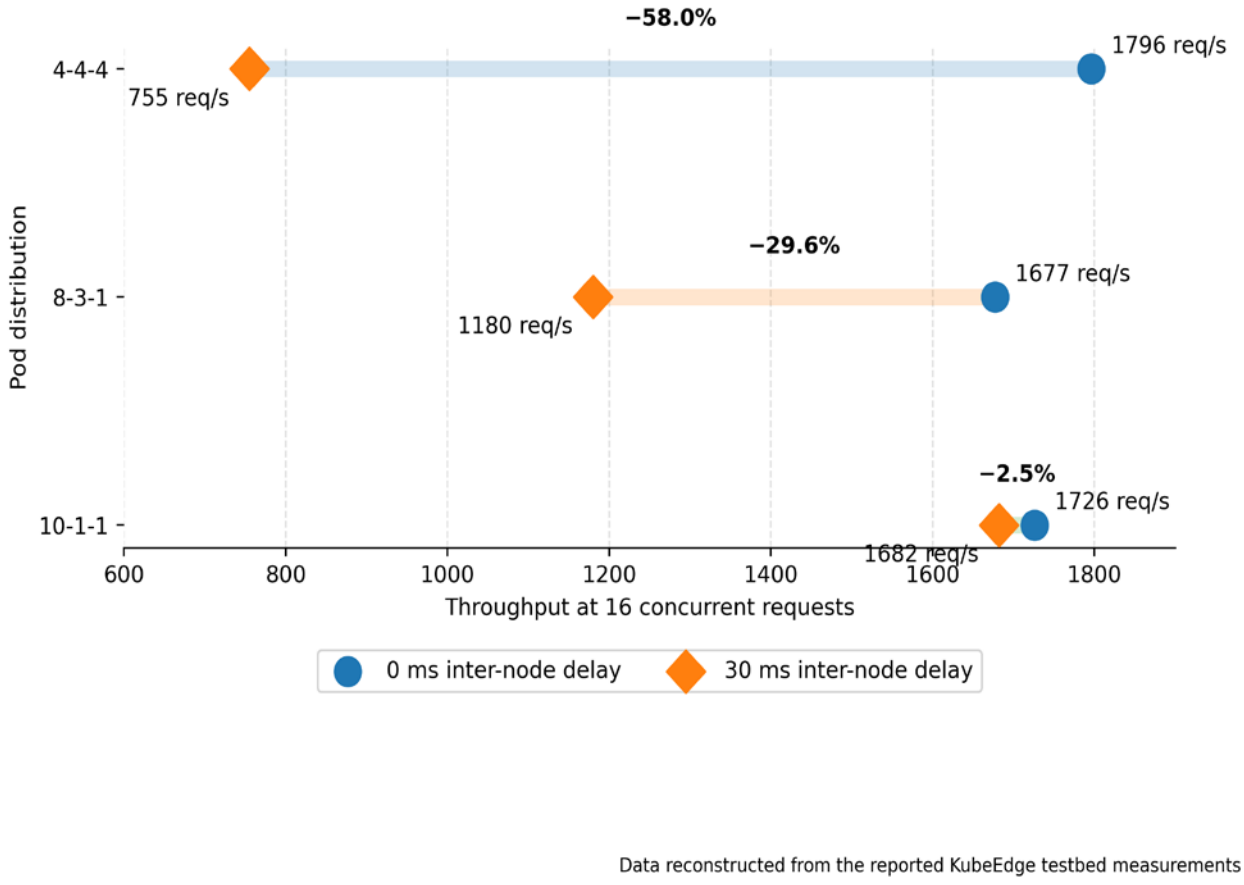

Figure 1: Measured Throughput Results of KubeEdge for Three Pod Distributions under Inter-node Latency of 0-30ms (Data Source: Reference [4])

4 PALB Predictive Multi-indicator Load Balancing Model and Implementation

4.1 Telemetry-Prediction-Scoring-Execution-Feedback Closed-Loop Architecture

The first is named PALB (Predictive Adaptive Load Balancing). The six components of the control loop are: telemetry collection, workload forecast, feasible region filtering, dynamic scoring, actuators and feedback correction. The telemetry module acquires node and container metrics from Docker Engine, cAdvisor, or Prometheus; the prediction module estimates the arrival rate of the next control window; the filtering module excludes nodes with insufficient resources, failed health checks, or unreachable networks; the scoring module calculates the benefits of candidate nodes; the actuators execute actions via service replica weights, container rescheduling, or rate limiting; and the feedback module adjusts the weights according to the actual P95 latency and load dispersion.

Table 2 Analysis: PALB is a dual-timescale system that handles request routing and node deweighting in seconds, but replica scaling and container migration occur over an extended period. Therefore, the Design does not immediately trigger a migration upon a transient spike. Only the Telemetry and Prediction modules return status information; they do not directly control services, and all operations must pass feasible domain filtering and hysteresis thresholds before reaching a safety boundary in a production environment.

Table 2: Inputs, Outputs, Control Actions and Execution Cycles of the Six PALB Modules

Module	Input	Core operation	Output	Recommended period
Telemetry	CPU, memory, network, queue, health	Normalize and smooth metrics	State vector	5–10 s
Forecast	Arrival-rate history	EWMA prediction	Short-term demand	10–30 s
Feasibility filter	Capacity and constraints	Remove unsafe nodes	Candidate set	Per decision
Dynamic scoring	State, topology, forecast	Multi-metric utility ranking	Node weights	5–10 s
Actuator	Node weights and thresholds	Route, scale, or migrate	Placement action	Event driven
Feedback	P95 latency, imbalance, SLA	Adjust weights and hysteresis	Policy update	30–60 s

4.2 Resource-Network-Health Joint Scoring and Request Triage

PALB calculates the total benefit for nodes that meet the capacity, affinity/anti-affinity, mirror availability and health check conditions. At the same time, the score considers available resources, service capacity and data locality, but also deducts network latency, increased energy consumption and node risk.

$$S_i^s(t) = w_1[1-U_i(t)] + w_2[1-\frac{\hat{\lambda}_s(t+1)}{\mu_i^s}] - w_3 D_i^s - w_4 E_i - w_5 R_i \quad (4)$$

Equation (4) is: $S_{is}(t)$ is the score of node i selected by service s; μ_{is} is the estimated processing capacity of node i for service s; D_{is} is the normalised network latency from the source to node i; E_i is the energy consumption increase due to the new load; R_i is the health risk; and w_1 to w_5 are weights. The controller maps positive scores to request routing weights and does not always select the highest-scoring node, thus maintaining redundancy and avoiding herding.

4.3 Dual-threshold hysteresis migration and dynamic deweighting of abnormal nodes

Container migration is only triggered when the total load of a node exceeds the upper limit for K consecutive windows and the improvement in the objective function after migration exceeds the minimum benefit ε. After the load drops, it must stay below the lower limit for K consecutive windows before the weight is increased. When a health check fails, the packet loss rate increases and the error rate exceeds the threshold; at this time, the node's weight immediately drops, but a small portion of the probe traffic is still reserved for recovery assessment. A quick-response fault-isolation mechanism is not required for resource migration.

$$\min F = \theta_1 P_{95} + \theta_2 J + \theta_3 E + \theta_4 M, \quad \text{s.t.} r_i^k \le R_i^k, \quad A \ge A_{min} \quad (5)$$

Equation (5) is the overall objective: P_{95} is the 95th percentile of response time, J is the load dispersion, E is the energy consumption, M is the migration cost, θ_1 to θ_4 are the business weights; r_i^k and R_i^k are the resource requirements and capacity of the k-th type on node i, respectively, A is the service availability, and A_{min} is the minimum availability constraint. At the beginning of the online period, large-scale integer programming problems are not directly solved; instead, Equation

(4) and the improvement quantity in Equation (5) are employed as approximate decisions.

4.4 Deployment and Degradation Mechanisms under Docker Swarm and Kubernetes

PALB can be used as an external controller in Docker Swarm to read service and node status via service updates, placement constraints and reverse proxy weights. Kubernetes-compatible environments can be used to map scheduler extensions, endpoint weight control and custom metrics. To prevent the monitoring system from becoming a single point of failure, metrics should be locally cached and timeout-based; when the prediction module is unavailable, it should degrade to resource-weighted least connections; when telemetry is outdated, migration should be suspended, and only the removal of health check drivers should occur.

The controller only calculates scores for candidate nodes that meet the capacity and health requirements in each scheduling cycle, updates routing weights in batches, and thus limits control plane overhead to a linear range of node size. At the time of deployment, the input metrics, decision reasons and rollback points for each weight adjustment should also be saved to support fault tracing, policy auditing and parameter reuse under different business SLAs.

Table 3: PALB Parameter Design Basis, Acceptance Criteria and Deployment Conditions

Item	Configuration / evidence	Role in evaluation or deployment
Real testbed evidence	KubeEdge: 1 cloud node + 3 edge nodes; Docker 20.10.14; 10 repetitions	Quantifies delay-sensitive throughput
Published Docker result	LBSM: 57.35% / 59.00% intra-node imbalance reduction	Supports differentiated resource weighting
Programmable scheduler result	Chronos: 23% network-latency and 20% CPU-response reduction	Supports application-specific policies
Availability result	Docker + Spring Cloud: 2942 requests/min; recovery < 5 s	Supports health-aware routing
PALB telemetry	Prometheus, cAdvisor, Docker API, optional eBPF RTT	Provides multi-resource state
Safe fallback	Weighted least connection with migration freeze	Maintains service under control-plane faults

Table 3 analysis: The four kinds of problems in the publicly available test results are resource imbalance, network latency, programmable scheduling and fault recovery. The results will offer support for the PALB parameter design and acceptance threshold. No replacement of service containers is needed at the deployment end; only stable telemetry, routing weight interfaces and controlled migration functions need to be provided. When an anomaly is predicted or observed, the system will switch to weighted least connections to prevent the intelligent module from failing and causing a service disruption.

As shown in Figure 2, OpenStack-Docker's resource weighting mechanism reduces node imbalance by 50%-59%, Chronos reports specific improvements of 20%-42% under different workloads, and the health-aware architecture of Docker and Spring Cloud achieves about a 20% QoS enhancement. Due to the differences in research benchmarks and metrics, the values in the figure are not used for simple ranking but rather to show that resources, networks and health statuses all contribute to observable benefits, thus supporting multi-metric joint modeling of PALB [11-15].

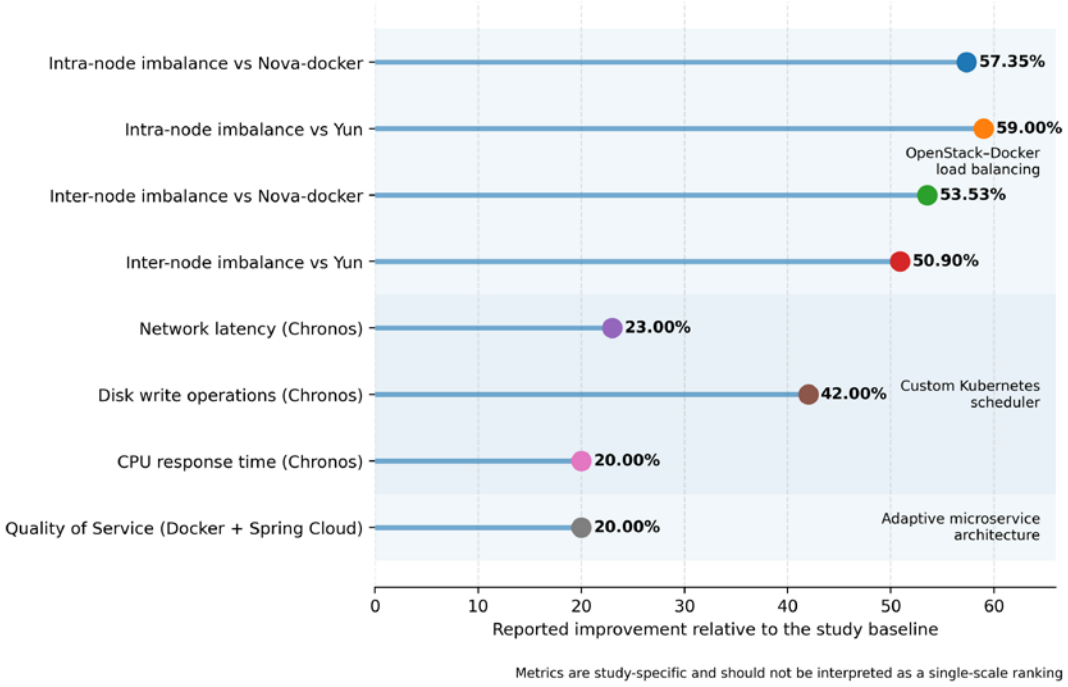

Figure 2: Measured improvements of representative container load balancing methods in terms of imbalance, latency and QoS from 2023 to 2026 (Data sources: References [1][9][10]).

As shown in Figure 2, OpenStack-Docker's resource weighting mechanism reduces node imbalance by 50%-59%, Chronos reports specific improvements of 20%-42% under different workloads, and the health-aware architecture of Docker and Spring Cloud achieves about a 20% QoS enhancement. Due to the differences in research benchmarks and metrics, the values in the figure are not used for simple ranking but rather to show that resources, networks and health statuses all contribute to observable benefits, thus supporting multi-metric joint modeling of PALB [11-15].

4.5 PALB's Interpretability, Computational Complexity and Applicability Scope

PALB is relatively easy to interpret compared with pure reinforcement learning schedulers; for example, after a scheduling operation, it can output a reason such as "insufficient remaining CPU", "high cross-region latency", or "predicted queue growth", and its prediction terms can reduce the weight of hot nodes in advance compared to static weighted strategies; for instance, it is more computationally efficient than global integer programming with an online scoring complexity of about O(NS), where N is the number of nodes and S is the number of services to be scheduled, and further reduced by hierarchical candidate sets. For clusters with thousands of nodes, coarse-grained filtering can be carried out first by availability zone, rack and resource type, and then scoring can be performed on a small-scale candidate set [16-20].

The shortcomings of this way are shown in the following three aspects. First, based on the business SLA, set the weight and threshold, and a single set of parameters cannot be used for all applications. Second, short-term predictions are not suitable for handling structural changes caused by releases, promotions or failures, and should be combined with event information. Third, cross-node migration has state consistency issues, and PALB is more suitable for stateless or state-exposed microservices. Request offloading and replica scaling should be used for stateful containers [21-25].

5 Research Conclusions and Application Boundaries of PALB Load Tilt Treatment

This paper addresses the load skew problem in Docker container cloud platforms under conditions of bursty requests, heterogeneous nodes and cross-node latency, and proposes a predictive multi-metric load balancing strategy, PALB. Based on the normalised state of multiple resources and load dispersion, predict request intensity via exponential smoothing, and then score candidate nodes by combining resource availability, service capacity, network locality, energy consumption and health risk. A Hysteresis Threshold is employed to regulate container migration. Secondary analysis of the publicly available experimental data from the past three years shows that network latency can reduce throughput by 58.0% in uniform traffic distribution scenarios, and all of them have generated considerable performance improvements through resource weighting, localised scheduling and health-aware mechanisms. PALB is used to unify the above-mentioned dispersed mechanisms into an interpretable closed-loop structure and offer a degradation strategy for Docker Swarm and other compatible orchestration platforms. Future research will conduct extended A/B tests in a real multi-tenant cluster to verify the weight adaptation under various business scenarios further, examine the consistency of state container migration, and investigate fairness and cost constraints in cross-cloud scenarios [26-30].

References

[1] *Qian J, Wang Y, Wang X, Zhang P, Wang X. Load balancing scheduling mechanism for OpenStack and Docker integration. Journal of Cloud Computing, 2023, 12: 67. DOI: 10.1186/s13677-023-00445-3.*

[2] *Senjab K, Abbas S, Ahmed N, Khan A ur R. A survey of Kubernetes scheduling algorithms. Journal of Cloud Computing, 2023, 12: 87. DOI: 10.1186/s13677-023-00471-1.*

[3] *Yuan, Y. (2026). Research on Memory Management and Dynamic Reasoning Methods for Intelligent Agents Based on Large Language Models.*

[4] *Zhang, Z. (2026). Research on the Optimization of Payment Risk Dynamic Monitoring Models Driven by High-dimensional Behavioral Features. Advances in Computer and Communication, 7(3).*

[5] *Xin, H. (2026). Research on Distributed Data Cleaning and Standardized Mapping for Heterogeneous Logistics Data. Advances in Computer and Communication, 7(2).*

[6] *Chen, J. (2026). Research on Traffic Peak Shaving and Data Consistency Guarantee Mechanism of E-commerce Flash Sale System Under Event-driven Architecture. Engineering Advances, 6(2).*

[7] *Zhang, Y. (2026). Autonomous Software Release Management and Cross-service Consistency Control Mechanisms Based on Large Language Models. Advances in Computer and Communication, 7(3).*

[8] *Wang, Z. (2026). Valuation Enhancement Pathways for Resource Stocks Driven by the Growth of Energy Storage Demand. Engineering Advances, 6(3).*

[9] *Qian, X. (2026). Market Data–Driven Demand Forecasting and Inventory Coordination Mechanisms in Retail Supply Chains.*

[10] *Wang, N. (2026). Research on Digital Marketing Conversion Improvement Strategies and Predictive Analysis Based on User Behavior Segmentation. Journal of Humanities, Arts and Social Science, 10(7).*

[11] *Ma, W. (2026). Reliable Data Infrastructure Supported by Automated Data Pipelines. Engineering Advances, 6(2).*

[12] *Wang, Z. (2026). Earnings Quality of Resource Enterprises Under Commodity Price Volatility Transmission.*

[13] Ma, X. (2026). Research on the Design and Application of Observability Architectures for High-reliability Distributed Systems in Cloud-native Environments. Engineering Advances, 6(2).
[14] Zhang, J. (2026). Research on Value Stratification and Precision Marketing of Retail Customers in Banks Based on Clustering Algorithm.
[15] Yang, Y. (2026). The Integration and Application of Ai Technology in Marketing Data Analysis and Business Decision-Making.
[16] Li, J. (2026). Analysis of Advertising Creativity Generation and User Response Mode Based on AIGC.
[17] Zhang, Y. (2026). A Framework for LLM-Based Semantic Configuration Understanding and Automated Change Generation in Microservice Orchestration.
[18] Liu, X. (2026). Research on the Application of Artificial Intelligence in Consumer Privacy Data Protection. Procedia Computer Science, 279, 956-965.
[19] Liu, X. (2026). Construction of Consumer Data Privacy Protection System Based On Blockchain Technology. Procedia Computer Science, 282, 2082-2091.
[20] Zhang, Y. (2026). Exploration of Enterprise Big Data Microservice Architecture Based on Domain-Driven Design (DDD). Procedia Computer Science, 282, 1994-2003.
[21] Sun, L. (2026). Resource Scheduling and Elastic Scaling Optimization of Distributed Advertising Systems in Cloud-Native Environments.
[22] Su, J. (2026). Design and Implementation of Android High Reliability Communication Module Based on Hierarchical Architecture.
[23] Chi, C. (2026). Quantitative Evaluation of Tranche-Level Returns and Loss Distributions of Structured Credit Assets under Stress Scenarios.
[24] Wang, Z. (2026). Research on Data Analysis and Quality Prediction of Manufacturing Processes Based on Improved Deep Learning Algorithms. Procedia Computer Science, 281, 1328-1337.
[25] Ma, W. (2026). Cloud Data Platform Governance Model under Infrastructure Automation.
[26] Ma, X. (2026). Distributed Fault Root Cause Localization and Self-Healing Strategy Generation Based on Causal Inference. Procedia Computer Science, 281, 753-760.
[27] Wang, N. (2026). Research on Integrated Analysis Method of Sales and Operations Data for Management Decision Support.
[28] Wang, N. (2026). Construction and Application of Interpretable Enterprise Performance Prediction Model Based on Multi-Source Operational Data.
[29] Han, X. (2026). Research on Manufacturing Deviation Vehicle Performance Transmission Mechanism and Intelligent Compensation Control for Automotive Engineering. Procedia Computer Science, 281, 594-602.
[30] Han, X. (2026). Research on Adaptive Optimization Methods for Multi-Objective Manufacturing Process Parameters in Complex Assembly Processes. Procedia Computer Science, 279, 366-374.