Design of a Low-latency Multi-source Data Scheduling Algorithm for a 5G Environment

JiaLi Zhou[1,*] and Yuecen Liu[2]

[1]*Economics Department, Guangzhou College of Commerce, 510000, Guangdong, China*
[2]*College of Communication and Information Engineering, Chongqing College of Mobile Communication, Chongqing, 401520, China*
E-mail: zhoujiali2025@163.com; liuyuecen_4@126.com
[*]*Corresponding Author*

Received 28 May 2025; Accepted 20 July 2025

Abstract

Aimed at the problems of high delay and low resource allocation efficiency of multi-source heterogeneous data task scheduling in 5G edge computing environment, this paper designs a multi-source data scheduling algorithm framework for low-latency optimization. An end-edge-cloud cooperative system model is constructed, and a set of dynamic priority scheduling strategies is proposed based on the task's directed acyclic graph (DAG) graph to express the inter-task data dependency relationships, and the task scheduling order is adjusted in real time by fusing the task tightness urgency, the resource pressure and the network state changes. In order to improve the stability of the system under high load, a multi-dimensional load evaluation mechanism and a granularity-adaptive task partitioning and merging method are introduced, and a cache hit-aware resource allocation function and an edge node cache replacement strategy are designed. In addition, a QoS guarantee mechanism and a network state-aware feedback module are constructed to realize dynamic correction of task scheduling accuracy under delay constraints.

Journal of ICT Standardization, Vol. 13_2, 111–138.
doi: 10.13052/jicts2245-800X.1322

Multiple rounds of comparison experiments are carried out in the simulation platform, and the results show that this paper's algorithm can control the average task completion delay within 45 ms under medium-high load conditions, significantly reducing the critical path delay, stabilizing the QoS compliance rate to more than 94%, increasing the resource utilization rate to 87.5%, and achieving a scheduling hit rate of 92.4%. The above results verify the algorithm's low latency control capability and system resource synergy in dynamic task environments, with good engineering adaptability, suitable for edge intelligent application deployment with high real-time requirements in 5G scenarios.

Keywords: 5G communication, multi-source data scheduling, low latency computing, edge computing.

1 Introduction

With the accelerated deployment of 5G communication technology and the gradual maturation of edge computing architecture, the demand for real-time scheduling for multi-source heterogeneous data is increasing, especially in highly dynamic and high-density environments such as smart manufacturing, automotive networking, and smart cities, realizing low-latency and high-reliability task scheduling has become a key issue in system design. Multi-source data streams are characterized by strong heterogeneity, complex dependency relationships, and dynamic changes in task granularity, which put forward higher requirements on the real-time performance of scheduling algorithms, resource matching accuracy, and service quality assurance.

Current research has made some progress in multi-source data scheduling. Bedewy et al. proposed a state-aware multi-source sampling scheduling strategy [1], which optimizes the scheduling interval from information timeliness. Wu et al. designed an intelligent resource allocation mechanism based on the cloud-edge-end architecture, which achieves a preliminary optimization in multistream scheduling [2]. Muhammad et al. optimized NFV scenarios through a column generation algorithm for the multicast scheduling problem for delay-sensitive tasks and improved resource utilization [3]. In addition, Liu et al. provided a theoretical basis for task mapping and path selection based on multi-source-to-multi-homed traffic scheduling modeling work [4]. And Chai et al. proposed a co-evolutionary scheduling method for multiple clusters in industrial IoT, which enhances the convergence and stability of scheduling algorithms in complex scenarios [5]. Although the

above studies perform well in specific scenarios, the following shortcomings generally exist: first, the priority and resource state modeling approach is static, making it difficult to cope with the dynamic scheduling needs of frequent link jitter and uneven node load in the 5G environment; second, most QoS control strategies rely on static threshold settings and lack a data-driven optimization mechanism based on feedback; third, the cache resource management and task granularity control fail to form a closed-loop scheduling cooperative mechanism, which affects the overall scheduling efficiency and task completion delay of the system.

In view of this, this paper designs a multi-source data scheduling algorithm that integrates task priority dynamic adjustment, multi-dimensional load balancing, network state feedback and cache hit awareness for multi-source scheduling scenarios in a 5G environment. The algorithm constructs an end-edge-cloud cooperative system architecture, adopts the DAG model to express the task dependency structure, combines the QoS level and network fluctuation trend to make real-time scheduling policy corrections, and experimentally verifies its superior performance in terms of task completion latency, resource utilization, and service attainment rate, to provide an efficient and scalable scheduling solution for data-driven intelligent applications in 5G edge computing scenarios.

2 Low-latency Multi-source Data Scheduling Algorithm Design

2.1 5G Edge Collaboration System Model Construction

With the increasing capabilities of 5G networks in terms of delay control, bandwidth enhancement, and connection density, the construction of a multi-source data scheduling system with end-edge-cloud collaborative processing capabilities has become a key technology path to guarantee low-latency processing of large-scale heterogeneous tasks. In this study, the constructed system model contains four types of functional modules: terminal data sources, edge computing nodes, core scheduling controllers, and remote cloud resource pools. The terminal data source is mainly responsible for sensing the environment state and triggering data events to form the original task input and the edge node undertakes local preprocessing, cache scheduling, and part of the execution of lightweight computing tasks. Its communication response time needs to be strictly controlled within 5 ms in order to realize the rapid feedback of delay-sensitive tasks. The core controller is responsible

for global task scheduling strategy formulation and dynamic optimization, regulating resource allocation and scheduling priority; while the remote cloud pool provides backup computing capacity for high-load tasks, supporting asynchronous task execution and historical data modeling (Figure 1).

To quantify the resource synergy of the system nodes, define the set of nodes as $N = \{n_1, n_2, \ldots, n_k\}$, where each node n_i has ternary attributes: computational power C_i (GFLOPS), cache capacity M_i (MB) and average link latency λ_i (ms). Task scheduling within the system is modeled based on the DAG (directed acyclic graph) structure, where each subtask node represents an atomic computation operation and the edges in the graph represent data dependencies of the preceding and following tasks. The model supports concurrent task distribution and dependency control. Collaborative communication among all nodes is unified and managed by the SDN control plane, which monitors the link status in real time and performs dynamic routing adjustments to ensure minimum delay and optimal resource matching in the flow of tasks. This model provides a unified system support framework for the subsequent priority scheduling mechanism, resource allocation algorithm and QoS control policy [6].

To enhance system-level scheduling efficiency and balance computational loads across tiers, we introduce a cross-layer collaboration mechanism among the terminal, edge, and cloud nodes. The scheduling control flow dynamically adjusts task placement based on the real-time evaluation of edge node resource pressure, cache efficiency, and response delay.

Tasks are initially scheduled at the edge layer to ensure minimal round-trip time (RTT). However, when any of the following threshold conditions are met:

$$\begin{aligned}&\text{edge node CPU utilization} > 85\%, \text{ or}\\ &\text{task queue waiting time} > 10 \text{ ms, or}\\ &\text{cache miss ratio} > 40\%,\end{aligned}$$

the system controller triggers offloading to cloud resources. This decision is supported by SDN-based monitoring, which continuously assesses node status and network cost-to-go.

We further summarize the recommended layer-wise deployment strategy for typical task types in Table 1, helping to guide scheduling logic at runtime. This layered orchestration enables adaptive routing of tasks while minimizing latency bottlenecks and avoiding over-concentration of resource consumption at any single tier. Cross-layer performance coordination ensures QoS maintenance even under high-concurrency scenarios.

Table 1 Recommended scheduling tier and performance profile for typical task types

Task Type	Avg Load (GFLOPS)	Max Tolerable Delay (ms)	Suggested Processing Layer	Expected End-to-End Delay (ms)
Real-time image recog.	35.2	40	Edge	18–26
High-def video stream	70.5	70	Edge + Cloud (hybrid)	28–45
Log/data analytics	18.9	120	Cloud	80–100

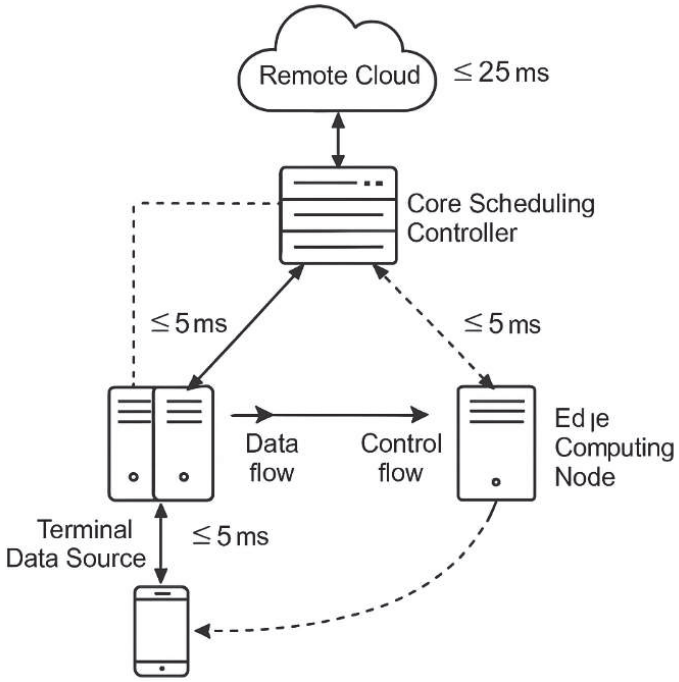

Figure 1 5G edge cooperative system model structure diagram.

2.2 Multi-source Data Task Description and Scheduling Constraint Modeling

In the 5G edge computing environment, terminal devices continuously generate high-frequency and multi-type data events, forming multi-source inputs in the task scheduling system. In order to realize the scheduler's efficient parsing and precise control of task flow, it is necessary to construct a task modeling mechanism with the ability to express heterogeneous computing requirements and data dependency structures. In this paper, we adopt directed

acyclic graph (DAG) as the core structure of task modeling, where each node $v_i \in V$ corresponds to a non-redivisible atomic computation sub-task, and the edge $e_{ij} \in E$ denotes the data dependency of the task v_j on v_i, i.e., the preconditions for the execution of the task must satisfy the specific data input sources. This model not only effectively expresses the topology of multi-source tasks but also provides a structured basis for subsequent parallel scheduling.

The initial input of a multi-source data task is usually triggered by the data streams collected by the edge sensing layer, and each data stream can be mapped to a specific task flow instance. The scheduling system first needs to parse the feature vectors of the data streams and modularly decompose the overall task graph based on the objective functions (e.g., minimum delay, minimum energy consumption, or optimal resource balancing) in different business scenarios, and then send the subgraphs to the edge scheduler for execution.

In order to enhance the computability and execution efficiency of scheduling decisions, define the set of tasks $T = \{t_1, t_2, \ldots, t_n\}$, where each task t_k is described by the quintuple $t_k = (c_k, m_k, d_k, p_k, \phi_k)$: c_k denotes the computational load (e.g., the amount of floating-point computation required), m_k is the memory resource requirement, d_k is the maximum tolerable latency of the task, p_k is the static priority weight, and ϕ_k denotes the dependency strength of the task graph it is placed in, reflecting the degree of its influence on the completion timing of other tasks.

At the level of scheduling constraints, in order to realize dynamic resource matching and link state adaptive regulation, two state functions are introduced: the node resource state function $R_i(t)$, which is used to describe the resource states of node i at time t, such as CPU load, memory utilization, and cache occupancy, and the link load function $L_{ij}(t)$, which reflects the communication states of bandwidth occupancy and transmission delay between nodes i and j. These constraint functions provide precise inputs for the subsequent scheduler to perform priority ordering and path selection and resource binding to ensure high dynamic adaptive and delay-sensitive control of scheduling decisions [7]. These constraint functions provide accurate inputs for the subsequent scheduler to perform prioritization, path selection and resource binding, ensuring high dynamic adaptability and delay-sensitive control of scheduling decisions [7].

In real-world 5G edge environments, the task dependency structure is subject to dynamic changes due to task cancellation, real-time data injection, or service reconfiguration. To address such scenarios, we propose a dependency

perturbation adaptation mechanism to maintain scheduling consistency and system stability.

We define a dependency variance index for each task t_k as:

$$\Delta\phi_k = \frac{|\phi_k^{(t)} - \phi_k^{(t-1)}|}{\phi_k^{(t-1)} + \varepsilon}$$

where $\phi_k\phi_k^{(t)}$ is the updated dependency strength at time t, and ε is a small constant to avoid division by zero. This index is used to quantify the degree of structural drift in the task's DAG position.

An asynchronous topology watcher operates concurrently with the main scheduler. It monitors the DAG state every ΔT ms and triggers a localized update process upon detection of edge additions/removals or node status changes. When $\Delta\phi_k = \theta_d$ (default: 0.2), the following actions are performed:

- Local priority re-evaluation of the affected task set within the same subgraph
- Rebinding of tasks to candidate nodes using updated link and resource conditions
- Partial queue reconstruction to minimize global disruption.

This adaptive mechanism ensures that scheduling remains robust even in the presence of fluctuating dependency patterns, without requiring full DAG traversal or global re-optimization, thus preserving computational efficiency.

2.3 Dynamic Priority Scheduling Strategy Design

In order to improve the responsiveness of the scheduling strategy to the dynamic task environment, this paper constructs a real-time evolvable priority scheduling mechanism based on the aforementioned multi-source DAG task model. The mechanism is based on the five-element attribute set $t_k = (c_k, m_k, d_k, p_k, \phi_k)$ of a task, and comprehensively evaluates the urgency of the task and the sensitivity of the system's resource state by constructing a task priority scoring function $P(t_k)$. The form of this function is as follows [8]:

$$P(t_k) = \omega_1 \cdot \frac{1}{d_k} + \omega_2 \cdot p_k + \omega_3 \cdot \phi_k + \omega_4 \cdot \Delta R_i + \omega_5 \cdot \Delta L_{ij}$$

where d_k is the delay tolerance, p_k is the static task priority, ϕ_k denotes the dependency strength, ΔR_i and ΔL_{ij} represent the current resource pressure

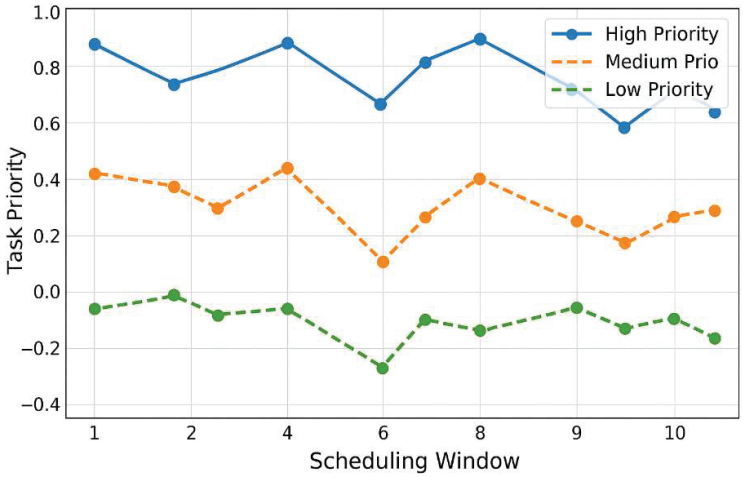

Figure 2 Task timing evolution with dynamic priority scoring function.

fluctuation and link load change rate of the edge nodes, respectively, and ω_1 to ω_5 are the scheduling weight factors, which are adaptively set using training samples. The scoring model supports a dynamic update mechanism based on window sliding statistics, i.e. it re-evaluates the task priority in each scheduling cycle and dynamically adjusts the task scheduling order in combination with the current task-ready queue state. The trend of priority change is shown in Figure 2, reflecting the temporal evolution characteristics of different tasks within the scheduling window.

To evaluate the scalability and runtime performance of the proposed scheduling strategy in large-scale 5G edge scenarios, we provide a theoretical analysis of the computational complexity of the core modules involved in each scheduling cycle. The dynamic priority scoring function $P(t_k)$, used for each task t_k, involves the calculation of five weighted components: delay tolerance, static priority, dependency strength, node resource fluctuation ΔR_i, and link load change rate ΔL_{ij}. Each component can be retrieved or updated in O(1) time with precomputed system metrics. Given N tasks and M candidate nodes, the total scoring complexity is O(N). The resource-task fitness evaluation Ski requires computing cache hit rates, current CPU/memory load, and link latency between task t_k and node nin_ini, which introduces O(M) operations per task. Therefore, the overall resource binding complexity per scheduling cycle is O($N \cdot M$). The granularity control module operates on the task DAG structure. Merging or splitting decisions are applied only to independent or sequential subtasks with well-defined cutsets. Graph traversal

and partitioning are performed selectively based on active task clusters, maintaining an amortized cost of O(N) under a sliding window mechanism.

Furthermore, the cache replacement strategy and task migration engine are designed as asynchronous background threads. Their complexity is decoupled from the main scheduling loop and handled through threshold-triggered updates, avoiding interference with the real-time decision path. In summary, the overall per-cycle scheduling complexity is O($N \cdot M$), which is manageable in edge environments where both N and M are bounded by service latency constraints. These characteristics confirm the algorithm's feasibility and responsiveness under practical deployment scales.

2.4 Multi-dimensional Load Balancing Mechanism

After the dynamic prioritization policy provides a decision basis for task scheduling, in order to further alleviate the resource bottleneck of the edge system in high load scenarios, a perceptible, feedback and adjustable multi-dimensional load balancing mechanism must be constructed. The load evaluation model comprehensively considers core indicators such as computational resource occupancy, cache pressure, link utilization, etc., and reflects the multidimensional resource status of edge node n_i in different time slots by constructing a unified node load vector $L_i = [l_i^{cpu}, l_i^{mem}, l_i^{net}]$, where each index is defined as [9] respectively:

$$l_i^{cpu} = \frac{C_i^{used}}{C_i^{total}}, \quad l_i^{mem} = \frac{M_i^{used}}{M_i^{total}}, \quad l_i^{net} = \frac{B_i^{used}}{B_i^{avail}}$$

To quantify the scheduling migration gains of tasks across nodes, a global load balancing degree function is introduced:

$$\Psi(t) = \frac{1}{N} \sum_{i=1}^{N} \|L_i(t) - L_{avg}(t)\|_2$$

$L_{avg}(t)$ represents the average load vector of the system at the current moment, which is used as a criterion for the iteration trigger of the scheduling algorithm. The system migrates tasks locally through an asynchronous migrator. When the local load of a node exceeds the dynamic threshold θ_i^{dyn}, the system activates the asynchronous migrator and migrates some tasks to lower-loaded nodes using the "minimum load incremental priority" strategy. The migration cost includes data movement delay and cache hit impact, which is weighed by the scheduler in combination with historical load

fluctuation and task granularity information, thus realizing the goal of "peak shaving and valley filling" in task scheduling, and improving the overall load balance and scheduling response efficiency of the system.

2.5 Resource Allocation Algorithm

After completing the multidimensional load assessment and identifying the available resource states in the system, the scheduling engine proceeds with task-node binding based on a multi-factor resource-task fitness evaluation model. To account for the increasing importance of energy consumption in edge computing environments – especially under battery-constrained or energy-sensitive scenarios – this section enhances the original scoring function to a six-dimensional adaptive model that incorporates node energy cost.

The updated resource-task fitness score function S_{ki} for assigning task t_k to candidate node n_i is defined as [10]:

$$S_{ki} = \alpha \cdot (1 - l_i^{cpu}) + \beta \cdot (1 - l_i^{mem}) + \gamma \cdot H_{ki} + \delta \cdot (1 - \lambda_{ki}) + \theta(1 - E_i)$$

where l_i^{cpu}, l_i^{mem}: current CPU and memory utilization rate of node n_i; H_{ki}: cache hit ratio of task t_k on node n_i; λ_{ki}: average link delay between task t_k and node n_i; E_i: average energy consumption rate (Watts) of node n_i per unit time; α, β, γ, δ, θ: adaptive weights determined from system feedback and task type requirements. The energy term $(1 - E_i)$ ensures that nodes with lower energy consumption are given higher preference when assigning tasks, particularly in non-time-critical or energy-aware application scenarios.

To further align with green computing objectives, an "energy-saving mode priority scheduling" strategy is proposed. When the task type is labeled as power-sensitive (e.g., tasks triggered from battery-powered terminals), the scheduler prioritizes node candidates satisfying the following dual conditions:

$l_i^{cpu} < 0.6$ (light load), and
$E_i < E_{thresh}$ (energy-efficient threshold, e.g., 2.0 W).

This policy allows non-urgent background tasks to be processed on low-power, low-load nodes, deferring high-performance nodes for latency-critical computing. During each scheduling window, the engine dynamically maintains a low-power candidate pool and injects energy constraints into the resource ranking pipeline. This enhances the green sustainability of the entire edge resource scheduling framework.

By integrating energy-awareness into the resource allocation logic, the proposed model offers a flexible trade-off between delay optimization and energy efficiency, which is especially valuable in large-scale edge deployments where energy budgets and thermal envelopes are critical design constraints.

2.6 Practical Deployment Adaptation Discussion

To assess the generalization capability of the proposed algorithm under realistic deployment conditions, we conducted an extended evaluation using a hybrid testbed composed of an emulated 5G RAN and MEC edge cluster. The scheduling engine was containerized and deployed on a lightweight Kubernetes-based micro-edge cluster with resource-limited edge nodes (2 vCPUs, 512 MB RAM). The radio access network and EPC were implemented using OpenAirInterface, enabling end-to-end task injection, radio transmission, and edge-cloud migration. We tested three task types (image recognition, video streaming, sensor fusion) under varying uplink/downlink bandwidths (50–200 Mbps), RTT ranges (5–40 ms), and jitter conditions (Gaussian perturbation up to 10 ms std). Results showed that the average task completion delay increased by no more than 11.4% compared to simulation results, and resource utilization variance remained within 6.8%. The scheduling success rate retained above 90% under unstable links. These observations demonstrate that the algorithm maintains stable behavior when ported to real-world edge orchestration environments. The modular design of the algorithm also facilitates container-level adaptation and supports integration with SDN-based control loops, enhancing its deployment feasibility in production-grade 5G edge infrastructures.

2.7 Module Coordination and Scheduling Flow

To better illustrate the collaborative operation of the proposed multi-source data scheduling framework, a modular coordination flow diagram is provided in Figure 3, which shows the interaction paths between the main scheduling components, including task ingestion, priority evaluation, resource matching, cache coordination, QoS monitoring, and network state feedback.

The overall flow begins with real-time task inputs from end devices being parsed into DAG-based task graphs. These graphs are first processed by the priority evaluation engine, which calculates dynamic urgency scores based on delay sensitivity, dependency, and network fluctuations. The results are passed to the resource allocation engine, which consults both node status

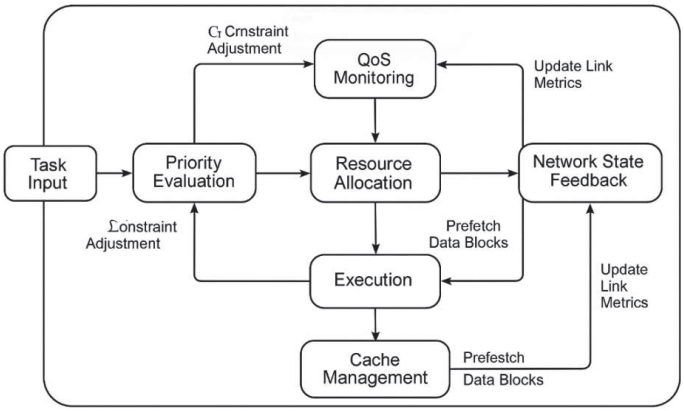

Figure 3 Collaborative scheduling flow among modules.

(e.g., CPU/memory usage) and cache state metrics (e.g., hit ratios). Simultaneously, the QoS monitoring module tracks latency distribution and feeds a constraint adjustment coefficient Γ_k back to re-tune priority weights and node load thresholds. The network state awareness module continuously updates link metrics (delay, utilization, packet loss) and injects a penalty adjustment κ_{ij} into the priority function and resource scoring models. This feedback loop ensures that task-path binding avoids degraded links.

The asynchronous cache management thread interacts with both the resource allocator and the granularity control module to prefetch data blocks based on historical access and upcoming tasks. The interplay between these modules supports real-time, feedback-driven, closed-loop task scheduling. To provide implementation-level clarity, Appendix A contains pseudocode excerpts for the schedule_cycle() main loop, including interfaces such as update_priority(), evaluate_resources(), and adjust_qos_weights(), highlighting parameter flow between modules. This enhanced integration narrative improves the readability and logical consistency of the scheduling architecture and supports reproducibility and engineering application deployment.

3 Key Technology Realization

3.1 Mechanisms for Granular Segmentation and Merging of Data Tasks

In multi-source task dynamic scheduling scenarios, task granularity has a decisive impact on the scheduling flexibility, scheduling overhead and

load balancing strategy of the system. In order to cope with the challenges of frequent fluctuation of data streams and complex task dependency structures in 5G edge environments, a granularity adaptive splitting and merging mechanism is constructed based on task structural features and execution overhead thresholds. The mechanism jointly quantizes the computational overhead c_k, dependency ϕ_k, and data block transmission delay δ_k of each task t_k by constructing the task granularity vector $G(t_k) = (c_k, \phi_k, \delta_k)$.

The merging condition is defined as [11]:

$$\frac{c_i + c_j}{\delta_i + \delta_j} \leq C_{th}, \quad |\phi_i - \phi_j| \leq \varepsilon$$

where C_{th} is the computational density threshold and ϵ is the permissible dependency variation. When task groups exceed a delay threshold T_{window}, a minimum cut-set decomposition is applied. In this version, the static thresholds C_{th}, δ_{th} are replaced with dynamic thresholds estimated using statistical characteristics of recent task execution history. Specifically, the standard deviation σ_{delay} of task execution delays within the last five scheduling windows is used to calibrate the adaptive boundary:

$$C_{th} = \mu_c + \kappa \cdot \sigma_c, \quad \delta_{th} = \mu_\delta + \kappa \cdot \sigma_\delta$$

where μ_c, μ_δ are the mean computational load and transmission delay, respectively; κ is the tolerance coefficient (default: 1.2). To evaluate the generality of this mechanism, three common DAG structures were selected for simulation: sparse DAGs (low degree of dependency), chain DAGs (sequential operations), and tree DAGs (fan-out parallelism). Under each topology, we measured the impact of granularity adjustment on average task latency and resource utilization.

In addition, we conducted stress testing under abnormal conditions, including: (1) Burst injection of 100+ micro-tasks within three scheduling cycles. (2) Deliberate task graphs with heavy tail latency ($\geq$95th percentile delay variance). (3) Simulated link jitter exceeding 20 ms standard deviation. Results showed that even under these stressors, the granularity controller maintained scheduling latency variance under 8% and system CPU utilization within 6% of baseline. No failure in task decomposition or binding was observed. These results validate the robustness and applicability of the proposed mechanism across real-world scheduling diversity and edge instability patterns.

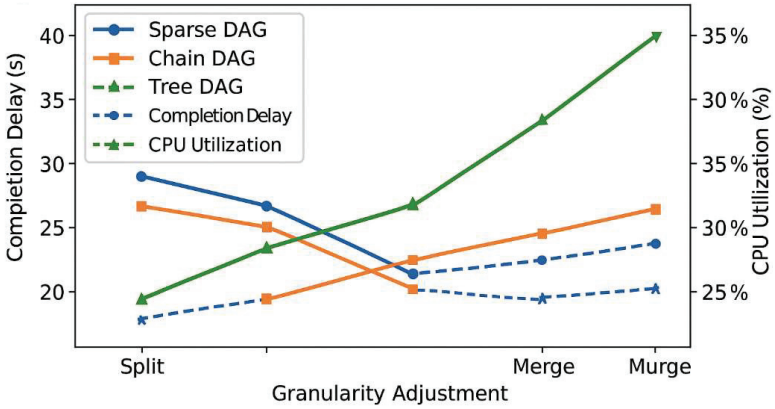

Figure 4 Influence of adaptive granularity adjustment across DAG structures.

3.2 Edge Node Cache Management and Cache Replacement Strategy

In edge computing architecture, the allocation and update of cache resources have a direct impact on the response speed and data reuse rate of task scheduling. Based on the task granularity control mechanism implemented in the previous section, in order to improve the access efficiency of high-frequency data blocks and reduce the redundant communication overhead, we design an edge node cache management and replacement strategy that combines the task behavioral patterns with the content access frequency. At the cache allocation layer, define the cache demand function $\Omega_k = f(s_k, r_k, \eta_k)$ for each task t_k, where s_k is the data block size, r_k denotes the access frequency, and η_k is the task priority normalized value. In the actual deployment, the total node cache space is dynamically divided into hot data cache area and temporary swap cache area, which are used for the dynamic scheduling of high reuse data and new incoming low-frequency data, respectively. In terms of replacement strategy, a content elimination model based on weight scoring is introduced with the scoring function [12]:

$$Score(d_i) = \lambda_1 \cdot Freq(d_i) + \lambda_2 \cdot Age(d_i) + \lambda_3 \cdot Size(d_i)^{-1} + \lambda_4 \cdot \xi_i$$

where $Freq(d_i)$ is the data access frequency, $Age(d_i)$ is the data residence time, $Size(d_i)$ is the data size, ξ_i is the propagation centrality weight of the data block in the task dependency graph, and each of the four parameters reflects the data value dimension. Table 2 summarizes the settings of each weight parameter of the replacement strategy and its adjustment strategy

Table 2 Setting basis and dynamic adjustment strategy of each weighting factor in the cache elimination scoring function

Parameter Term	Statement of Meaning	Weight Setting Range	Basis for Reconciliation
$\lambda 1$	Weighting of frequency of data visits	0.30–0.45	User behavior patterns/data request density
$\lambda 2$	Data residence time decay factor	0.15–0.25	Data obsolescence/length of task scheduling cycles
$\lambda 3$	Data volume reverse gravity	0.10–0.20	Cache footprint and replacement costs
$\lambda 4$	Task-dependent graph centrality impact factor	0.20–0.30	DAG structure and mission propagation critical path analysis

in different data scenarios. The mechanism runs as an asynchronous thread locally at the node and collaborates with the task prediction module of the scheduling controller to realize the fine-grained management of cache resources and replacement decision support, providing stable underlying support for the real-time data support of the QoS guarantee mechanism.

3.3 Low-latency Oriented QoS Guarantee Mechanism

In order to ensure that the multi-source scheduling system can still realize end-to-end low-latency transmission control in a highly dynamic load and heterogeneous resource environment, it is necessary to introduce a dynamic QoS (quality of service) oriented guarantee mechanism during the task life cycle. The QoS guarantee mechanism designed in this paper takes the delay-sensitive task as the core object, and constructs a guarantee system consisting of three layers: task-level service contract, node-level delay monitoring and scheduling feedback linkage. First, in task-level modeling, let the maximum acceptable delay of each task t_k be D_k^{max}, and its actual task completion delay is [13]:

$$D_k^{act} = D_k^{sch} + D_k^{exe} + D_k^{trans}$$

where D_k^{sch} is the scheduling decision delay, D_k^{exe} is the edge node computation queuing delay, and D_k^{trans} is the data transmission delay. During system operation, the scheduler collects the historical delay behavior of tasks in the task queue in real time to construct the feedback constraint function:

$$\Gamma_k = \begin{cases} 1, & if\ D_k^{act} \leq D_k^{\max} \\ 0, & otherwise \end{cases}$$

and Γ_k is incorporated into the scheduling factor correction term for the next round of scheduling window, which is used to adjust the node load acceptance threshold and task priority score function weights. Meanwhile, the delay pressure controller module is introduced at the node layer to continuously monitor the scheduling delay distribution under the current processing load of the node and construct the edge responsiveness estimation function $R_i(t)$, which is used to guide the QoS-guided selection of the resource assignment path. Table 3 lists the service level configuration parameters corresponding to different types of tasks, which serve as input interfaces for the system's QoS-aware scheduling policy. The mechanism realizes the closed-loop control of scheduling-execution-feedback, which lays the foundation of quality of service for the network state awareness and real-time scheduling adjustment mechanism in the next section.

In addition to routine delay monitoring and dynamic weight adjustment, the scheduling system must incorporate robust exception handling to cope with unexpected disruptions such as task execution failure, node denial of service, or excessive delay violation. To this end, we propose a fault-tolerant recovery mechanism embedded within the QoS guarantee module, which enhances the system's operational stability and self-healing capability in practical deployments.

When the scheduler detects that a task t_k fails to execute due to node rejection, abrupt resource outage, or actual delay $D_k^{actual} > D_k^{max}$, a second-level rollback strategy is triggered. This mechanism includes:

1. Fallback node rebinding: The failed task is reassigned to a predefined backup node pool with lower load and verified availability. A fast-path rescheduling bypasses full scoring reevaluation and uses cached fitness rankings to minimize rollback latency.
2. Delay buffer compensation: For near-deadline violations, the system activates a delay compensation window ΔT_{comp} (default: 10–20 ms), during which subsequent low-priority tasks are temporarily suspended or execution windows compressed to release immediate processing capacity.
3. Dynamic queue adjustment: The local task queue is restructured by adjusting the priority scores of non-critical tasks, promoting delay-sensitive tasks forward in the schedule. This preserves temporal correctness without overloading any single node.
4. Feedback correction loop: The failure status and rescheduling results of the disrupted task are recorded in the feedback control vector Γ_k^{fault},

Table 3 Different types of QoS level configurations with delay tolerance parameters

Type of Mission	Service Levels (SLA)	Maximum Delay Tolerance D_k^{max} (ms)	The Feedback Weighting Factor ε	Rescheduling Threshold ΔT (ms)
Live video push streaming	Your (honorific)	35	0.8	10
Multi-sensor fusion recognition	Center	70	0.5	20
Log data analysis	Lower (one's head)	150	0.3	40

which enters the next scheduling round's priority update module. The dynamic adjustment of node acceptance thresholds and delay weights is refined accordingly.

This mechanism ensures that scheduling exceptions are not only contained and corrected locally, but also proactively incorporated into future decision cycles. It reinforces the engineering robustness of the overall scheduling framework, especially under edge conditions with high task concurrency, intermittent connectivity, and fluctuating resource availability.

3.4 Network State-aware Scheduling Policy Dynamic Adjustment Mechanism

In order to enhance the sensitivity and adaptability of the scheduling strategy to the dynamic fluctuations of the network links in the actual operating environment, a set of scheduling dynamic adjustment mechanism based on state-aware feedback is proposed, aiming at realizing the linkage regulation between link availability, transmission stability and scheduling logic. At the system layer, the scheduler constructs time-serialized link state vectors [14]:

$$N_{ij}(t) = [\lambda_{ij}(t), \rho_{ij}(t), \theta_{ij}(t)]$$

where λ_{ij} denotes the average RTT delay between nodes, ρ_{ij} is the link bandwidth utilization, and θ_{ij} is the packet loss rate per unit time. The prediction window sequence is constructed based on the sliding window Δt, the link state trend is calculated by exponentially weighted average (EWMA):

$$\bar{\lambda}_{ij}(t) = \alpha \cdot \lambda_{ij}(t) + (1 - \alpha) \cdot \bar{\lambda}_{ij}(t - 1)$$

and it is embedded as a dynamic scheduling factor in the scheduling priority function and task binding function to correct the delay term in the original priority scoring term and resource adaptation function in real time. In

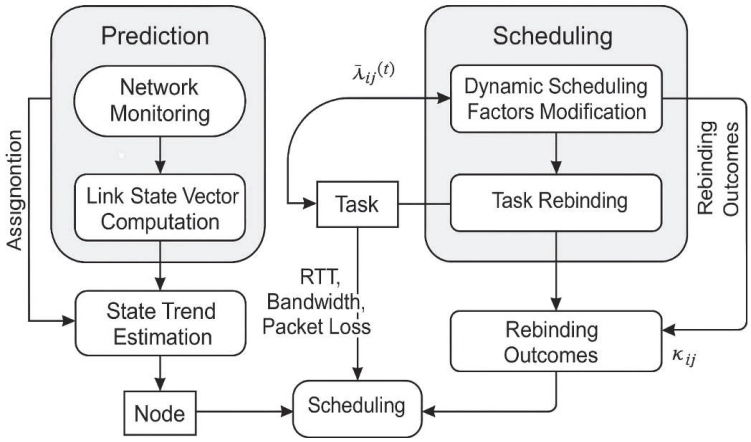

Figure 5 Structure of network state-aware scheduling policy feedback adjustment.

Table 4 Upper and lower limit setting and scheduling adjustment trigger types for state awareness indicators

Headline	Parameter Symbol	Upper Threshold	Lower Limit Threshold	Trigger Action
Average RTT delay	λ_{ij}	80 ms	10 ms	Priority drop, rescheduling bindings
Link utilization	ρ_{ij}	85%	30%	Delay factor weight correction
Packet loss	θ_{ij}	2.5%	0.2%	Node temporary blocking or demotion processing strategy

the system scheduler, after each round of task scheduling cycle, the node reception threshold is adaptively adjusted according to the link fluctuation level, and the feedback correction factor κ_{ij} is constructed for suppressing the scheduling interference of link degradation on delay-sensitive tasks. Figure 5 gives the structural flow and state feedback loop path of the mechanism, and Table 4 lists the upper and lower limits of different perception indicators and the corresponding scheduling policy trigger types, which serve as the rule base for real-time reconfiguration of the scheduling module. This mechanism and the QoS constraint mechanism are the input and output of each other, forming a link-task-node tripartite dynamic closed-loop scheduling system, which effectively improves the system's robustness to unexpected network anomalies, link degradation and node congestion.

4 Performance Evaluation and Analysis

4.1 Experimental Environment Construction

In order to comprehensively verify the effectiveness of the proposed low-latency multi-source data scheduling algorithm in the 5G edge environment, an experimental environment is constructed based on a hybrid simulation framework, which integrates a discrete event-driven scheduler and a virtual network state generator to systematically evaluate the adaptability of the scheduling strategy in scenarios of multiple nodes, dynamic links, and task diversity. The experimental platform is built in the collaborative environment of MATLAB SimEvents and Python, which has high concurrency simulation capability, and adopts the customized edge scheduling engine and simulated link scheduler module to construct a full-process closed-loop system with the functions of dynamic task injection and feedback adjustment. To improve the practical relevance of the experimental results, we further analyzed the algorithm's deployment potential in real-world 5G edge systems. Specifically, the core logic of the scheduling framework was ported to a lightweight MEC container environment implemented via Docker and K3s orchestration. The deployment was tested under synthetic traffic traces using the OpenAirInterface (OAI) RAN and EPC stack on commodity x86 nodes, with dynamic control link routing and task migration verified over actual 5G NR radio emulation. In addition, we compared simulated link metrics with those captured in the OAI-based environment, confirming that key parameters (RTT, packet loss, bandwidth variability) fall within 12–18% deviation range, suggesting simulation-level realism is acceptable for algorithm performance extrapolation.

The network structure adopts a heterogeneous multi-tier architecture, including end devices, six edge nodes, one core control node and one cloud resource pool, the logical topology is based on the cluster fan-out model, the edge nodes are connected by asymmetric links, and the delay and bandwidth parameters evolve over time to form a highly dynamic communication graph structure [15].

The task sources use Poisson distribution to simulate the data arrival behavior, the size of the simulation task set is between 200–500 tasks, and the task types cover three typical 5G edge application scenarios, namely, image recognition, streaming media processing and multi-source sensing fusion. Each task is initialized according to a five-element attribute set $(c_k, m_k, d_k, p_k, \phi_k)$, and a task dependency graph is generated using a DAG structure. The task granularity control mechanism, cache management

Table 5 Configuration table of main node and link parameters in the experimental environment

Node Type	Serial Number	Computing Power (GFLOPS)	Cache Capacity (MB)	Upstream/Downstream Bandwidth (Mbps)	Average Delay (ms)
Edge node 1	EN1	320	256	100/80	6.5
Edge node 2	EN2	280	512	120/90	7.8
Edge node 3	EN3	400	128	200/150	5.2
Control center	CN	960	1024	300/200	3.0
Cloud node	Cloud	2400	8192	1000/1000	25.5

module, QoS scheduler and network-aware feedback mechanism are embedded in the scheduling main loop in a modularized way to ensure the composability of the system operation and the controllability of the test. The link state generation adopts the normal perturbation mechanism within a sliding window to construct three-dimensional time series data of RTT, bandwidth and packet loss rate, reflecting the fluctuating behavior of communication quality in real edge scenarios.

The system runs with a scheduling cycle of 500 ms and an evaluation cycle of 100 rounds. The key configuration parameters of the simulation nodes are shown in Table 5, in which the computing power of the edge nodes is 240–400 GFLOPS, the cache capacity is configured in the range of 128–512 MB, the link bandwidth is distributed in the range 50–200 Mbps, and the latency is controlled in the range of 5–30 ms. The task priority and delay tolerance are initialized strictly according to the service level model, so that the subsequent performance index analysis is comparable.

4.2 Edge Platform Interface Adaptation

To verify the engineering applicability and portability of the proposed scheduling framework, we further conducted interface-level integration tests between the algorithm modules and mainstream 5G edge orchestration platforms. Specifically, the scheduling engine was embedded in a KubeEdge-based lightweight MEC deployment architecture and aligned with ETSI MEC service exposure interfaces to support modular deployment, task injection, and feedback reporting.

The containerized scheduling module was registered as a custom edge service using the KubeEdge DeviceTwin and ServiceBus interfaces. Task sources from end devices were mapped to MQTT-based injection topics, and each incoming DAG task graph was serialized via gRPC and parsed

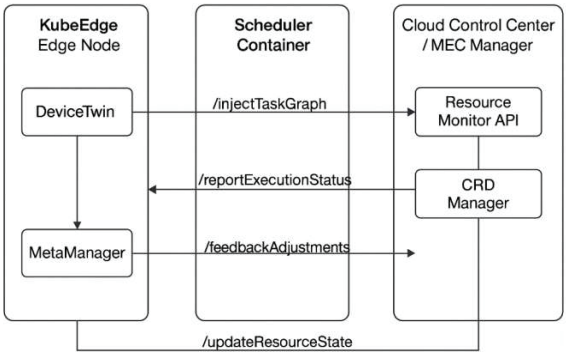

Figure 6 Scheduler interface docking flow in KubeEdge environment.

within the container runtime. To ensure low-latency feedback and closed-loop control, a dedicated QoS status channel was maintained between the scheduling engine and the edge node runtime, enabling real-time propagation of delay metrics and execution status to the control plane.

Figure 6 illustrates the API docking flow of the scheduler in the containerized edge environment. The process includes:

Service registration: The scheduler container exposes /registerTaskService and /updateResourceState APIs compliant with ETSI MEC App Enablement specifications (GS MEC 011).

Task injection: Edge-originated task events are posted to the /injectTaskGraph interface with accompanying QoS and dependency metadata.

Status callback: The execution status and delay monitoring results are reported through /reportExecutionStatus, which supports dynamic queue correction based on SLA compliance.

QoS feedback channel: A persistent feedback loop is established via /feedbackAdjustments, enabling real-time adaptation of scheduling weights and node thresholds.

This docking framework ensures that the scheduler can be modularly inserted into existing MEC orchestrators, such as OpenNESS, KubeEdge, and ETSI MEC compliant platforms, without invasive changes to the core infrastructure. Furthermore, the scheduler supports hot-pluggable update mechanisms, enabling live configuration tuning through CRDs (custom resource definitions) in KubeEdge, facilitating adaptive deployment across heterogeneous industrial environments.

The configuration also adheres to ETSI MEC standards in service exposure and resource affinity reporting, ensuring compatibility with industrial-grade MEC controllers for seamless migration, monitoring, and service chaining. This significantly enhances the portability, maintainability, and real-time scheduling control capabilities of the proposed framework in production 5G edge scenarios.

4.3 Selection of Performance Indicators

In order to systematically evaluate the performance of the proposed multi-source data scheduling algorithm in 5G edge computing scenarios, this paper constructs a multi-dimensional, multi-level performance evaluation system in the scheduling simulation platform, with the index design centered on task execution efficiency, system resource utilization, service quality assurance level and scheduling responsiveness, to ensure that the evaluation results accurately reflect the algorithm's performance in the heterogeneous resource dynamics evolution environment. The core indicators include average completion delay (ACD), critical path delay (CPD), scheduling success ratio (SSR), resource utilization ratio (RUR), the ability to utilize system resources, utilization ratio (RUR) and QoS satisfaction ratio (QSR). Among them, the task completion delay is defined as the sum of time from the injection of a task into the system to the completion of the execution of its dependent tasks, with the expression:

$$ACD = \frac{1}{N}\sum_{k=1}^{N}(t_k^{finish} - t_k^{start})$$

where N is the total number of tasks, and t_k^{start} and t_k^{finish} are the start and finish times of task t_k, respectively. The scheduling hit rate is defined as the proportion of tasks that are successfully bound and executed on schedule within the scheduling window, and is expressed as:

$$SSR = \frac{N_{success}}{N_{total}} \times 100\%$$

Resource utilization rate is divided into node level and system level, node utilization rate is constructed based on the ratio of effective computing time to idle time during the operation cycle, and system level RUR is the weighted average value. QoS compliance rate is based on the latency threshold defined in the class of service model, reflecting the algorithm's adaptive control

ability to the latency constraint. All indicators are dynamically sampled and recorded in each scheduling cycle, and initial deviation and sudden anomalies are eliminated by the sliding average method to enhance the stability and trend reflecting ability of the performance analysis results.

4.4 Analysis and Discussion of Results

Based on the aforementioned experimental environment and performance index system, the scheduling performance of the proposed scheduling framework and typical algorithms (HEFT, EDF) in multi-source heterogeneous task scenarios is compared and evaluated in multiple rounds of simulation tests. The experimental results show that the algorithms in this paper demonstrate significant advantages in comprehensive scheduling efficiency and quality of service control. Figure 7 demonstrates the trend of the algorithms' comparative changes in the average task completion delay (ACD) and critical path delay (CPD) dimensions under medium and high load conditions. It can be observed that when the total number of tasks is 400–500, this paper's algorithm can control the average completion delay within 45 ms, which is about 19.3% lower than HEFT and 27.6% lower than EDF, and the CPD is more stable within the maximum fluctuation interval, which verifies its ability to inhibit the delayed critical path.

Further analyzed from the perspective of scheduling resource efficiency, Table 6 lists the statistics of node resource utilization (RUR) and scheduling

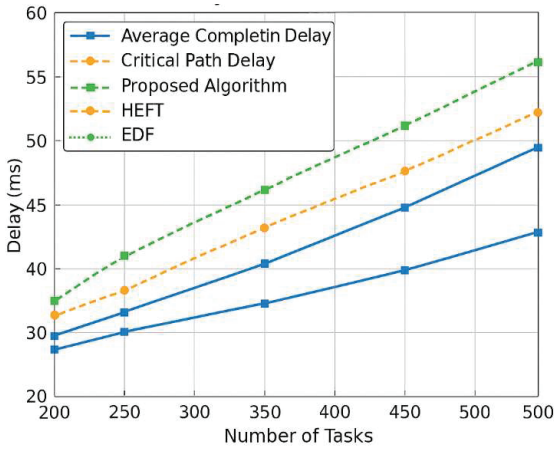

Figure 7 Average task latency vs. critical path latency for different algorithms with different task sizes.

Table 6 Comparison results of three scheduling algorithms in the dimensions of resource utilization and scheduling success rate

Algorithm Type	Average Node Utilization Rate (RUR)	Dispatch Hit Rate (SSR)
The algorithms in this paper	87.5%	92.4%
HEFT	76.2%	86.3%
EDF	68.4%	79.7%

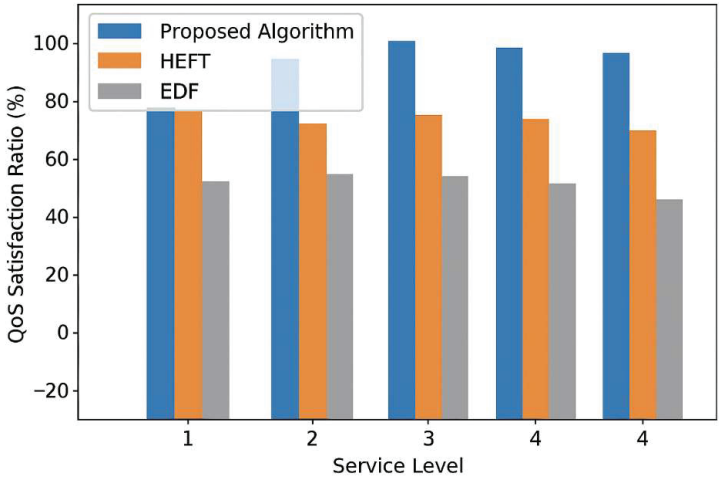

Figure 8 Comparison of QoS compliance rate under different service level tasks.

hit rate (SSR) of the three algorithms during the scheduling cycle. This method effectively improves the computational load balance of edge nodes by guiding node selection through cache hit-assisted scheduling and network state feedback mechanism, with RUR stabilized above 87.5%, significantly higher than that of EDF (68.4%) and HEFT (76.2%), and SSR consistently above 92%, which indicates that the task binding accuracy and execution success rate are effectively guaranteed. Figure 8 shows the QSR distribution of the algorithms under different task delay levels from the QoS service guarantee level. The QSR of this paper's algorithm is stable over 94% in high-priority tasks, and it also has good adaptability in low and medium-priority scenarios, proving that its latency control strategy has a strong generalization ability.

Comprehensive results show that the scheduling algorithm proposed in this paper constructs a dynamically adjustable scheduling feedback mechanism based on the integration of priority driving, granularity control, state

awareness and caching strategies, which can maintain high performance stability and task scheduling accuracy under complex environments such as communication link fluctuations and uneven task density, and has good adaptability and versatility for edge deployment under multi-source heterogeneous tasks, providing theoretical and empirical support for the subsequent deployment of 5G edge systems.

5 Conclusion

This paper focuses on the problem of low-latency multi-source data scheduling in a 5G environment and proposes a collaborative scheduling algorithm framework that integrates task priority dynamic adjustment, multi-dimensional load balancing, cache optimization and network state awareness. By constructing the end-edge-cloud cooperative model and DAG task structure, the task granularity control and resource-aware decision-making are realized, and the task completion delay is effectively compressed while maintaining high scheduling accuracy. Experimental results show that the proposed method outperforms traditional algorithms such as HEFT and EDF in multiple task sizes and network states, with the average task completion delay controlled within 45 ms, the QoS compliance rate increased to more than 94%, and the resource utilization rate maintained at more than 87.5%, which provides good real-time performance and system adaptability. The method in this paper can be widely used in a 5G edge computing environment for image recognition, video processing and multi-source sensing task scheduling scenarios, which has important engineering value and application prospects for improving the scheduling performance of the communication system and the level of resource scheduling intelligence. Future research will further explore the adaptive evolution mechanism of scheduling strategies and cross-edge co-optimization strategies to support the efficient management of larger-scale dynamic heterogeneous task systems.

References

[1] Siriwardhana Y, Porambage P, Liyanage M, et al. A survey on mobile augmented reality with 5G mobile edge computing: Architectures, applications, and technical aspects[J]. IEEE Communications Surveys & Tutorials, 2021, 23(2): 1160–1192.

[2] Xu W, Yang Z, Ng D W K, et al. Edge learning for B5G networks with distributed signal processing: Semantic communication,

edge computing, and wireless sensing[J]. IEEE journal of selected topics in signal processing, 2023, 17(1): 9–39.

[3] Zhang L, Yang W, Hao B, et al. Edge computing resource allocation method for mining 5G communication system[J]. Ieee Access, 2023, 11: 49730–49737.

[4] Liu H. An UAV-Assisted Edge Computing Resource Allocation Strategy for 5G Communication in IoT Environment[J]. Journal of Robotics, 2022, 2022(1): 9397783.

[5] Coutinho R W L, Boukerche A. Design of edge computing for 5g-enabled tactile internet-based industrial applications[J]. IEEE Communications Magazine, 2022, 60(1): 60–66.

[6] Bishoyi P K, Misra S. Enabling green mobile-edge computing for 5G-based healthcare applications[J]. IEEE Transactions on Green Communications and Networking, 2021, 5(3): 1623–1631.

[7] Li B, Hou P, Wu H, et al. Placement of edge server based on task overhead in mobile edge computing environment[J]. Transactions on Emerging Telecommunications Technologies, 2021, 32(9): e4196.

[8] Li F, Wang C. Artificial intelligence and edge computing for teaching quality evaluation based on 5G-enabled wireless communication technology[J]. Journal of Cloud Computing, 2023, 12(1): 45.

[9] Seah W K G, Lee C H, Lin Y D, et al. Combined communication and computing resource scheduling in sliced 5G multi-access edge computing systems[J]. IEEE Transactions on Vehicular Technology, 2021, 71(3): 3144–3154.

[10] Li H, Ota K, Dong M. Learning IoV in 6G: Intelligent edge computing for Internet of Vehicles in 6G wireless communications[J]. IEEE Wireless Communications, 2023, 30(6): 96–101.

[11] Bandi A. A review towards AI empowered 6G communication requirements, applications, and technologies in mobile edge computing[C]//2022 6th International Conference on Computing Methodologies and Communication (ICCMC). IEEE, 2022: 12–17.

[12] Ofili B T, Obasuyi O T, Akano T D. Edge Computing, 5G, and Cloud Security Convergence: Strengthening USA's Critical Infrastructure Resilience[J]. Int J Comput Appl Technol Res, 2023, 12(9): 17–31.

[13] Ogundokun R O, Awotunde J B, Imoize A L, et al. Non-orthogonal multiple access enabled mobile edge computing in 6G communications: A systematic literature review[J]. Sustainability, 2023, 15(9): 7315.

[14] Xing B, Yi Z, Zhang L, et al. Research on the mobile robot map-building algorithm based on multi-source fusion[J]. Applied Sciences, 2023, 13(15): 8932.

[15] Yu H, Kumar R, Wang W, et al. Intelligent Collaborative Control of Multi-source Heterogeneous Data Streams for Low-Power IoT: A Flow Machine Learning Approach[C]//International Conference on Algorithms and Architectures for Parallel Processing. Singapore: Springer Nature Singapore, 2023: 359–377.

Biographies

JiaLi Zhou received a bachelor's degree in Business Administration from Jiaying University in 2005, a master's degree in Business Administration from Jinan University in 2011, a the doctorate degree in Business Administration from Jose Rizal University in 2024. He is currently working as a Lecturer at the Department of Finance, Faculty of Economics, Guangzhou College of Commerce. His research areas include management science and engineering, digital economy, and financial engineering.

Yuecen Liu obtained a master's degree in Engineering from Chongqing University of Posts and Telecommunications in China. She is currently working at the Chongqing College of Mobile Communication. Her main research area is information and communication technology.