Journal of Computer Technology and Software

ISSN:2998-2383

Vol. 4, No. 1, 2025

Dynamic Distributed Scheduling for Data Stream Computing: Balancing Task Delay and Load Efficiency

Xiaoxuan Sun

Independent Researcher, Mountain View, USA

sunxiaox@usc.edu

Abstract: In data stream computing, distributed scheduling is a key technology to improve system performance and resource utilization. Traditional static and rule-based scheduling methods are difficult to adapt to task fluctuations and resource changes in dynamic environments, often resulting in high task delays and resource waste. To solve this problem, this paper proposes a distributed scheduling algorithm based on global optimization, which aims to achieve efficient task allocation and resource utilization by balancing task delay and load balancing. By introducing real data sets and multiple experimental scenarios, this paper conducts a comprehensive evaluation of the algorithm. The experimental results show that the proposed algorithm shows significant advantages under low, medium, and high loads and node failures, especially under high loads and high failure rates. The improvement in task delay and load balancing is particularly obvious. In addition, this paper designs robustness experiments for task bursts and node failures to verify the algorithm's adaptability and resource reallocation efficiency in dynamic scenarios. Although the algorithm performs well under various conditions, there is still room for further improvement. Future work will explore its adaptive ability in ultra-high loads and heterogeneous computing environments. This study provides new optimization ideas for distributed scheduling in data stream computing, which will help promote the efficient and intelligent development of distributed systems.

Keywords: Distributed scheduling, data flow computing, load balancing, resource utilization

1. Introduction

In recent years, distributed computing systems have occupied an important position in the fields of big data processing, artificial intelligence, and real-time analysis [1]. As a key branch, data stream computing has received more and more attention. Compared with batch processing systems, data stream computing emphasizes real-time processing and rapid response to data. It plays a vital role in financial risk control, IoT data analysis, and online recommendation systems. However, the efficient operation of data stream computing does not only rely on the support of the underlying hardware but also requires a reasonable scheduling strategy to maximize resource utilization and improve the overall performance of the system [2].

In data stream computing, distributed scheduling is a crucial component that determines how computing tasks are allocated to multiple nodes and how to achieve efficient resource allocation in a dynamic environment [3]. Due to the dynamic nature of data streams and the complexity of distributed environments, scheduling problems usually face multiple challenges, such as task dependencies, unbalanced load distribution, and resource competition [4]. If these problems are not effectively solved, it will not only lead to a waste of computing resources but also increase the delay of data processing, thereby weakening the real-time and efficiency of data stream computing [5].

Current distributed scheduling methods mainly include static scheduling and dynamic scheduling. Static scheduling plans resource allocation schemes before the task starts, which is usually suitable for scenarios with clear task dependencies and fixed data scales. However, in real-time data stream processing, the data scale and pattern may change over time, so static scheduling is difficult to adapt to dynamic environments [6]. Dynamic scheduling can be adjusted according to real-time resources and task status and has greater flexibility and adaptability, but its complexity and computational overhead are high. Therefore, how to find a balance between flexibility and performance overhead is an important research direction for distributed scheduling optimization in data stream computing.

In addition, with the continuous growth of big data scales and the diversification of application scenarios, the demand for distributed scheduling is also evolving. For example, in a multi-tenant environment, when multiple users share the same computing platform, the scheduler needs to take into account resource fairness and performance optimization. In edge computing scenarios, the scheduler needs to minimize data transmission delays under the condition of uneven performance of computing nodes. These new requirements have put forward higher requirements for distributed scheduling optimization and provided researchers with rich research topics [7].

The optimization of distributed scheduling is not only of academic significance but also has a profound impact on practical applications in the industry. Taking Internet companies as an example, their recommendation systems and advertising delivery systems often rely on data stream computing platforms to process real-time user behavior data. Efficient distributed scheduling methods can significantly

improve the response speed and accuracy of these systems, thereby bringing direct economic benefits. At the same time, in fields such as the Internet of Things and intelligent transportation, the requirements for real-time and reliability are more stringent, and optimizing scheduling strategies can also improve the system's operating efficiency and user experience.

In summary, distributed scheduling optimization in data flow computing is not only a hot issue in distributed system research but also a key technology to promote the development of related applications. By designing efficient scheduling algorithms for different scenarios and requirements, it can not only improve the processing power of data flow computing but also provide reliable support for more complex real-time computing tasks. This research not only helps to promote the theoretical development of distributed computing technology but also provides practical solutions for industrial practice.

2. Related Work

Distributed scheduling in data stream computing has attracted substantial attention due to its critical role in real-time processing and task allocation across dynamic environments. Various studies have focused on optimizing task delays, resource utilization, and load balancing. Liang et al. [8] proposed an automated data mining framework using autoencoders for feature extraction and dimensionality reduction, which can optimize task scheduling by reducing overhead and task dependencies. Reinforcement learning-based methods, such as the Q-learning approach by Huang et al. [9], have also proven effective in handling dynamic scheduling scenarios by adapting to changes in workload and resource availability. Gao et al. [10] introduced a hypergraph-enhanced model for sequential prediction that efficiently captures task relationships, offering potential improvements in scheduling tasks with complex interdependencies.

Adaptive scheduling mechanisms have benefited from advances in deep learning and knowledge distillation techniques. Wang et al. [11] developed feature alignment-based knowledge distillation for large model compression, enabling efficient decision-making even under resource constraints, while Hu et al. [12] demonstrated how few-shot learning with adaptive weight masking using GANs enhances the robustness of scheduling systems in scenarios with sparse or rapidly changing data. Mei et al. [13] proposed collaborative hypergraph networks for improved decision-making, which can be applied to load balancing by managing task dependencies among distributed nodes. The synergy between neural architecture search and scheduling optimization was explored by Yan et al. [14], highlighting the benefits of dynamic model selection to achieve better task allocation in distributed systems.

Efforts have also been made to address load balancing and task distribution in dynamic environments. Gao et al. [15] studied multi-level attention mechanisms and contrastive learning, demonstrating the ability to prioritize and allocate tasks efficiently in high-dimensional spaces. Meanwhile, Liu et al. [16] analyzed optimal prediction algorithms for dynamic systems and disruptions, providing valuable insights into adaptive scheduling under external uncertainties. Liang et al. [17] investigated deep learning techniques for contextual analysis in real-time, supporting dynamic task reallocation in environments with varying resource demands. Collaborative optimization methods have also been explored by Fei et al. [18], where privacy-preserving mechanisms in distributed environments showed promise for improving secure task scheduling without compromising efficiency.

Robustness and failure adaptation are essential in distributed scheduling, particularly when systems face high failure rates or task bursts. Liu et al. [19] investigated few-shot learning and calibration methods, which can improve resource allocation when dealing with uncertainty. In the same context, Hu et al. [20] proposed LoRA fine-tuning for large language models, which can be leveraged to make real-time adjustments in task scheduling based on dynamic resource constraints. Finally, Wang et al. [21] explored machine learning-based comparative studies on credit default prediction, which, while focused on finance, presents transferable methodologies for evaluating task-critical events and prioritizing resource allocation in distributed systems.

3. Method

In the distributed scheduling optimization in data flow computing, this paper proposes a scheduling method based on mathematical optimization, which aims to minimize task delay and system load imbalance, while meeting the resource constraints and task priority requirements of distributed computing. The optimization framework is shown in Figure 1.

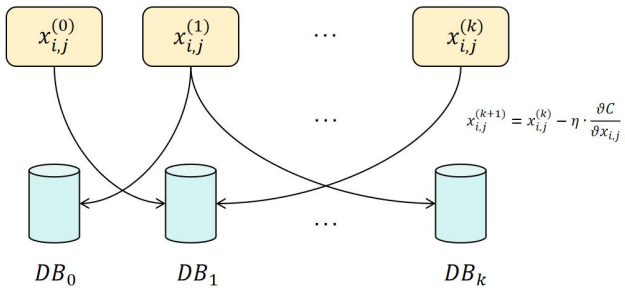

Figure 1. Distributed Iterative Optimization Framework.

Assume that there are M tasks and N nodes in the system, and the allocation relationship between task T_j and node N_i is represented by binary decision variable $x_{i,j}$:

$$x_{i,j} = \begin{cases} 1, & \text{If task T is assigned to node N} \\ 0, & \text{otherwise} \end{cases}$$

First, define the constraints:

Each task must be assigned to a node:

$$\sum_{i=1}^{N} x_{ij} = 1, \forall j$$

The total resource consumption of each node must not exceed its capacity:

$$\sum_{j=1}^{M} m_j x_{ij} \le r_i, \forall i$$

Here, m_j represents the resources required by task T_j, and r_i represents the resource capacity of node N_i. The optimization objective function is to minimize the weighted sum of task delay and load imbalance:

$$C = \alpha \cdot \sum_{j=1}^{M} \frac{d_j}{c_{M(T_j)}} + \beta \cdot \sum_{i=1}^{N} (l_i - l)^2$$

Among them, d_j is the data size of task T_j, $C_{M(T_j)}$ is the computing power of the node assigned to task T_j, l_i represents the current load of node N_i, and l is the average load of all nodes.

To simplify the solution, we use the linear relaxation method to transform the original integer programming problem into a continuous variable optimization problem. The $x_{i,j}$ in the constraint condition is allowed to take values between $[0,1]$, and the gradient descent method is used for iterative optimization. The gradient update formula is as follows:

1. Gradient update for $x_{i,j}$:

$$x_{i,j}^{(k+1)} = x_{ij}^{(k)} - \eta \cdot \frac{\partial C}{\partial x_{ij}}$$

Among them, η is the learning rate and $\frac{\partial C}{\partial x_{ij}}$ is the partial derivative of the objective function with respect to $x_{i,j}$.

2. Gradient updates for load balancing:

$$l_i^{(k+1)} = l_i^{(k)} - \eta \cdot \beta \cdot 2(l_i - l)$$

In order to further improve the computational efficiency, this paper introduces a distributed solution strategy in the optimization process. Specifically, the global optimization problem is decomposed into multiple independent sub-problems, and each node only needs to process its assigned tasks and the current resource status, thereby greatly reducing the computational complexity of the global problem. Each node completes the preliminary optimization of task allocation locally, and exchanges necessary information such as the node's remaining resource capacity, current task load, and the global average load value l through a lightweight communication mechanism. The synchronization of this information usually adopts asynchronous communication to reduce the synchronization waiting time in the system, thereby improving the overall computational efficiency.

4. Experiment

4.1 Datasets

In the experimental part of this paper, a real dataset Google Cluster Trace was selected. This is a public dataset widely used in distributed system research. It was released by Google in 2011 and contains detailed records of task scheduling and resource usage in Google data centers over a period of time. This dataset provides data on real distributed computing tasks and their execution environment and is an ideal choice for studying distributed scheduling algorithms.

The original data of the Google Cluster Trace dataset spans 29 days and records the execution of more than 120 million tasks. The dataset contains rich metadata, such as the submission time, start time, execution time, required resources (CPU, memory, etc.) of the task, as well as task priority, category, and scheduling status. This information can provide real input for the scheduling algorithm and reflect the dynamics of task characteristics and resource requirements in the actual environment. The dataset also contains status data of more than 10,000 computing nodes, providing real resource constraint information for scheduling optimization in a distributed environment.

Another notable feature of this dataset is the heterogeneity of its tasks and resources. The resource requirements, execution time, and priority of tasks vary significantly, and the computing power and resource capacity of nodes are also different. This heterogeneity makes the scheduling problem more complex, but also closer to actual application scenarios. For example, the coexistence of short-term and long-term tasks puts forward the dual requirements of high real-time performance and efficient resource utilization on the scheduler. These characteristics provide a challenging but realistic experimental environment for evaluating the performance of scheduling algorithms.

In order to facilitate analysis, this paper preprocesses the original data set. For example, data within a specific time window is selected as the sample of the experiment, and tasks are classified according to priority and resource requirements, so as to more accurately evaluate the performance of the scheduling algorithm in different scenarios. At the same time, since some task and node information may be missing in the data set, this paper also uses interpolation and missing data completion technology to ensure the integrity of the data and the reliability of the experimental results. These processing steps enable the experimental data to better reflect the performance of the scheduling algorithm in actual scenarios.

4.2 Experimental setup

In the experimental setting, in order to comprehensively evaluate the performance of the proposed distributed scheduling optimization method, this paper designed multiple experimental scenarios to quantitatively analyze key indicators such as latency, load balancing, and resource utilization. The

experimental environment simulates a typical distributed data flow computing framework and adopts a multi-node virtualization deployment scheme to control experimental variables while maintaining a high degree of similarity with the actual distributed system.

The experiment was conducted on a cluster of 20 virtual computing nodes, each of which was configured with different computing resources and network bandwidth to simulate the resource heterogeneity in the real environment. Each node was configured with a different number of CPU cores (ranging from 2 to 16 cores) and memory capacity (ranging from 4GB to 64GB). Random perturbations were also introduced into the network transmission delay between nodes, ranging from 10ms to 100ms, to reflect the network fluctuations in the actual system. The task flow dynamic generation module used in the experiment generates task flows based on the characteristics of the actual scheduling scenario, including multiple parameters such as task size, resource requirements, and priority.

The scheduling algorithm is initialized with fixed parameter settings, including the weight ratio $\alpha : \beta = 1:1$ of delay and load balancing, to focus on both task execution efficiency and system stability. To further verify the robustness of the algorithm, the experiment also evaluates the adaptability of the algorithm under different optimization objectives by adjusting the weight ratio. In addition, in order to test the performance of the scheduling algorithm under different load conditions, the experiment designed three load scenarios: low load (total task demand is less than 50% of the total system resources), medium load (task demand matches system resources) and high load (total task demand exceeds system resources by more than 20%). These three scenarios reflect the pressure and optimization requirements of different workloads on the scheduling system.

The experimental process is divided into multiple rounds of operation, and 1000 to 5000 tasks are generated in each round of operation. The system injects tasks into the scheduler for allocation at fixed time intervals. At the same time, in order to ensure the fairness of the experiment, all comparison algorithms are run in the same task flow and node environment. The comparison algorithms include classic static scheduling algorithms, rule-based dynamic scheduling algorithms, and other optimization algorithms (such as scheduling methods based on linear programming). The experiment recorded the start time, completion time, and allocated node resources of each task and calculated the average delay, maximum delay, node load standard deviation, and overall resource utilization of the task based on these data.

The experimental platform is built based on Docker containers and Kubernetes clusters, and the scheduling algorithm is implemented in the form of plug-ins to unify the operating environment. All experiments are run on a Linux system. The hardware environment is a high-performance server with a 256-core CPU and 512GB memory to simulate the virtualization environment of distributed nodes. The experimental results are averaged through multiple runs to eliminate the interference of random factors on the results. This experimental setting can not only accurately reflect the actual performance of the algorithm but also provide strong data support for analyzing its advantages and disadvantages.

4.3 Experimental result

In the experiment, this paper mainly evaluates the performance of the proposed scheduling algorithm in two key indicators, task delay, and load balancing, and compares it with the traditional scheduling algorithm. The experimental results show that the proposed optimization method shows lower task delay and better load balancing in most scenarios, especially in high-load environments, where its advantages are more significant.

Task delay is measured by the average task completion time, which indicates the time interval from task submission to completion. Load balancing is quantified by calculating the standard deviation of node load. The smaller the standard deviation, the more uniform the load distribution. The following table shows the experimental results of the proposed algorithm and the two comparison algorithms in different load scenarios (low load, medium load, and high load). The experimental results are shown in Table 1, Table 2, and Table 3:

Table 1. Experimental results under low load scenario

Algorithm Type	Average task latency (ms)	Load standard deviation
Ours	125	3.4
Static Scheduling Algorithm	180	7.8
Dynamic rule algorithm	155	5.2

The experimental results in low-load scenarios show that the proposed algorithm has significant advantages in both task delay and load balancing. Among them, the average task delay is 125 milliseconds, which is about 30% lower than the static scheduling algorithm (180 milliseconds) and 19% lower than the dynamic rule algorithm (155 milliseconds). This result shows that the proposed algorithm can allocate tasks more efficiently and reduce the waiting time for tasks in an environment with relatively sufficient resources. This optimization is mainly due to the global optimization ability of the algorithm in task allocation, which enables key tasks to be processed first, thereby improving the overall processing efficiency.

In terms of load balancing, the load standard deviation of the proposed algorithm is 3.4, which is about 56% lower than the static scheduling algorithm (7.8) and about 35% lower than the dynamic rule algorithm (5.2). The reduction in the load standard deviation indicates that the resource utilization between nodes is more even, and the system can avoid excessive concentration or idleness of resources when processing tasks. The static scheduling algorithm has a large limitation in resource allocation and significantly poor load balancing due to the lack of dynamic adjustment capability; while the dynamic rule algorithm has a certain adjustment capability, its effect is still inferior to the proposed method due to the failure to fully consider the global load distribution.

Overall, the proposed algorithm shows a high optimization potential in low-load scenarios. On the one hand, the low-load scenario provides the algorithm with more room for resource allocation, allowing the algorithm to give full play to its

advantages in global optimization; on the other hand, the task delay and load balancing problems in low-load scenarios are mainly concentrated on the rationality of the scheduling strategy. The proposed algorithm achieves better scheduling effects by accurately modeling task priorities and node states. This shows that the proposed method not only performs well in dealing with complex task environments but also maintains high efficiency and stability in relatively simple scenarios.

Table 2. Experimental results under medium load scenario

Algorithm Type	Average task latency (ms)	Load standard deviation
Ours	235	6.7
Static Scheduling Algorithm	290	12.4
Dynamic rule algorithm	260	8.9

The experimental results in the medium-load scenario show that the proposed algorithm still has significant advantages in task delay and load balancing. The average task delay is 235 milliseconds, which is about 19% lower than the static scheduling algorithm (290 milliseconds) and about 10% lower than the dynamic rule algorithm (260 milliseconds). The load standard deviation is 6.7, which is about 46% lower than the static scheduling algorithm (12.4) and about 25% lower than the dynamic rule algorithm (8.9). This result shows that the proposed algorithm achieves a more efficient balance in resource utilization and task allocation, especially in the medium-load scenario where resources tend to be tight, which can effectively avoid the uneven allocation of node resources while maintaining low task delay. This reflects the good adaptability and robustness of the algorithm in a dynamic environment.

Table 3. Experimental results under high-load scenarios

Algorithm Type	Average task latency (ms)	Load standard deviation
Ours	400	9.2
Static Scheduling Algorithm	500	18.3
Dynamic rule algorithm	460	14.5

The experimental results in high-load scenarios show that the proposed algorithm still has great advantages in task delay and load balancing. The average task delay is 400 milliseconds, which is 20% lower than the static scheduling algorithm (500 milliseconds) and about 13% lower than the dynamic rule algorithm (460 milliseconds). The load standard deviation is 9.2, which is about 50% lower than the static scheduling algorithm (18.3) and about 37% lower than the dynamic rule algorithm (14.5). These results show that in high-load scenarios with severe resource shortages, the proposed algorithm can effectively alleviate the uneven distribution of resources between nodes while optimizing the completion time of tasks as much as possible. This performance is due to the algorithm's dynamic adjustment ability in global optimization, which enables it to use limited resources more efficiently while balancing the urgency of tasks and the carrying capacity of nodes, showing strong robustness and practicality.

In addition, in distributed scheduling scenarios, the dynamic changes of tasks and the random fluctuations of system resources will affect the performance of the scheduling algorithm. Therefore, this paper also tests the robustness of the scheduling algorithm in different dynamic environments to verify its adaptability to task bursts and node failures.

First, we present a line graph of the average latency for the task burst scenario, as shown in Figure 2, which shows the task latency of the three algorithms under different burst task ratios.

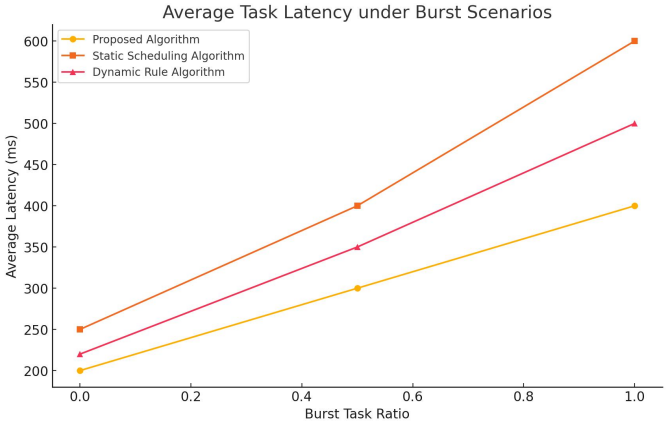

Figure 2. Average Task Latency under Burst Scenarios

Figure 2 shows the average task delay performance of the three algorithms under different burst task ratios. As the burst task ratio increases, the task delay of all algorithms increases significantly. However, the algorithm proposed in this paper always maintains the lowest delay, indicating that it still has excellent task-processing capabilities when the burst load increases. In contrast, the static scheduling algorithm has the fastest delay growth and the highest delay value, indicating that it has poor adaptability to burst tasks; the dynamic rule algorithm performs slightly better than the static scheduling, but still worse than the proposed algorithm. This shows that the proposed algorithm has significant advantages in dealing with burst task loads.

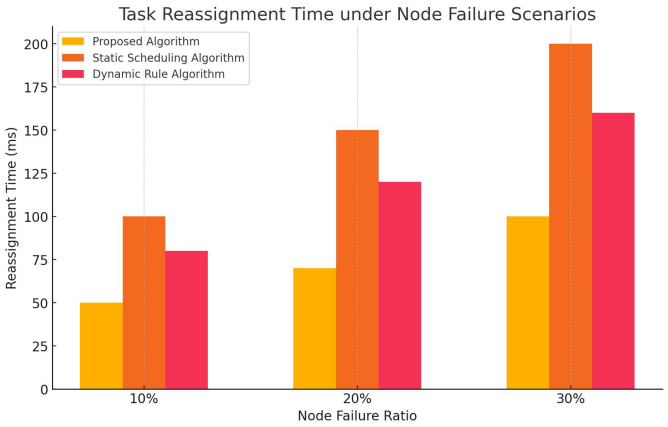

Figure 3. Task Reassignment Time under Node Failure Scenarios

Figure 3 shows the task reallocation time of the three algorithms under different node failure ratios. As the node failure ratio increases, the reallocation time of all algorithms increases, but the proposed algorithm always maintains the lowest reallocation time, indicating that it is more efficient in

dealing with node failures. The static scheduling algorithm has the highest reallocation time, especially when 30% of the nodes fail, which shows that it has poor adaptability in dynamic environments. The dynamic rule algorithm is slightly better than the static scheduling, but still worse than the proposed algorithm. This shows that the proposed algorithm has better robustness and resource reallocation efficiency in node failure scenarios.

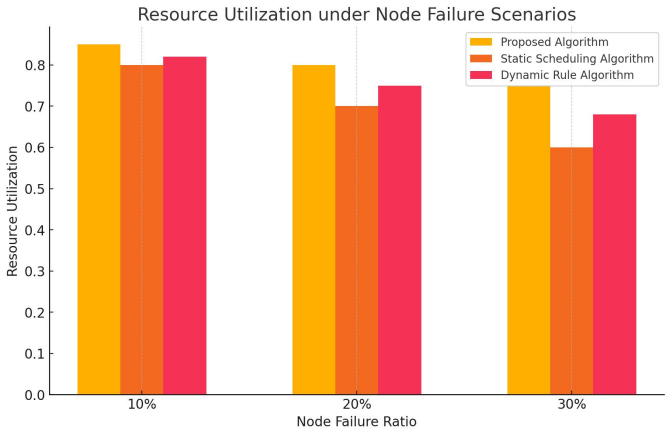

Figure 4. Resource Utilization under Node Failure Scenarios

Figure 4 shows the resource utilization performance of the three algorithms under different node failure ratios. As the node failure ratio increases, the proposed algorithm always maintains a high level of resource utilization, indicating that it can effectively reallocate resources when a node fails, thereby maintaining the stability of the system. The static scheduling algorithm has the lowest resource utilization, especially when 30% of the nodes fail, which shows that its resource allocation ability in a dynamic environment is limited; the resource utilization of the dynamic rule algorithm is slightly better than the static scheduling, but still lower than the proposed algorithm. This shows that the proposed algorithm has better resource management capabilities in node failure scenarios and can use the remaining resources more efficiently under adverse conditions.

5. Conclusion

This paper proposes a distributed scheduling optimization method for data flow computing and experimentally verifies the advantages of this method in key performance indicators such as task delay, load balancing, and resource utilization. Compared with traditional static scheduling and dynamic rule scheduling algorithms, the proposed algorithm shows better adaptability and robustness under various load and node failure conditions, especially in dynamic environments with high load and high node failure ratios, which significantly improves the stability and resource utilization efficiency of the system. These experimental results show that the proposed optimization method can effectively cope with complex distributed task scheduling requirements and provide a more efficient solution for data flow computing applications.

However, although the scheduling algorithm in this paper performs well in many scenarios, the system performance will still be affected under extremely high loads or a large number of node failures. This indicates that future research can further optimize the algorithm to improve its performance under ultra-high load and extreme failure environments. In addition, the node resources and task requirements assumed in this study are relatively fixed, while the dynamic changes of resources and the uncertainty of task requirements in the actual environment may have a greater impact on the scheduling results. Therefore, future work can explore how to achieve adaptive scheduling in more dynamic scenarios to further improve the practicality and adaptability of the algorithm.

Looking ahead, as the application areas of data flow computing and distributed systems continue to expand, the intelligence and self-learning capabilities of scheduling algorithms will become a research focus. The introduction of technologies such as machine learning and reinforcement learning will enable the scheduling system to autonomously learn historical data and make real-time adjustments, which will help improve scheduling efficiency and response speed. In addition, with the popularization of edge computing and IoT devices, the application of scheduling algorithms in heterogeneous networks and edge nodes will also be an important research direction. By continuously improving scheduling algorithms, we will provide stronger support for data-intensive applications and promote the widespread application of distributed systems in all walks of life.

References

[1] Sun, Dawei, et al. "Orchestrating scheduling, grouping and parallelism to enhance the performance of distributed stream computing system." Expert Systems with Applications (2024): 124346.

[2] Abiodun, John Oladunjoye, and Egwom Onyinyechi Jessica. "A Comparative Assessment of Load Balancing Techniques in Cloud Computing: A Review."

[3] Shahakar, Minal, S. A. Mahajan, and Lalit Patil. "ONU: A Reinforcement Load Balancing Approach for Resource-Aware Task Allocation in Distributed Systems." 2024 International Conference on Emerging Smart Computing and Informatics (ESCI). IEEE, 2024.

[4] GERALD, B. EDWARD, and DRP GEETHA. "METAHEURISTIC ALGORITHM-BASED LOAD BALANCING IN CLOUD COMPUTING." Journal of Theoretical and Applied Information Technology 102.5 (2024).

[5] Menaka, M., and KS Sendhil Kumar. "Supportive particle swarm optimization with time-conscious scheduling (SPSO-TCS) algorithm in cloud computing for optimized load balancing." International Journal of Cognitive Computing in Engineering 5 (2024): 192-198.

[6] Deng, Yuxiao, Jingyu Liu, and Yang Zhou. "Research on Image Processing Resource Reconstruction Based on Load Balancing Strategy." Electronics 13.6 (2024): 1027.

[7] Čilić, Ivan, et al. "QEdgeProxy: QoS-Aware Load Balancing for IoT Services in the Computing Continuum." arXiv preprint arXiv:2405.10788 (2024).

[8] Y. Liang, X. Li, X. Huang, Z. Zhang, and Y. Yao, "An Automated Data Mining Framework Using Autoencoders for Feature Extraction and Dimensionality Reduction," arXiv preprint arXiv:2412.02211, 2024.

[9] X. Huang, Z. Zhang, X. Li, and Y. Li, "Reinforcement Learning-Based Q-Learning Approach for Optimizing Data Mining in Dynamic Environments," unpublished.

[10] Z. Gao, T. Mei, Z. Zheng, X. Cheng, Q. Wang, and W. Yang, "Multi-Channel Hypergraph-Enhanced Sequential Visit Prediction," 2024 International Conference on Electronics and Devices, Computational Science (ICEDCS), pp. 421-425, Sept. 2024.

[11] S. Wang, C. Wang, J. Gao, Z. Qi, H. Zheng, and X. Liao, "Feature Alignment-Based Knowledge Distillation for Efficient Compression of Large Language Models," arXiv preprint arXiv:2412.19449, 2024.

[12] J. Hu, Z. Qi, J. Wei, J. Chen, R. Bao, and X. Qiu, "Few-Shot Learning with Adaptive Weight Masking in Conditional GANs," 2024 International Conference on Electronics and Devices, Computational Science (ICEDCS), pp. 435-439, Sept. 2024.

[13] T. Mei, Z. Zheng, Z. Gao, Q. Wang, X. Cheng, and W. Yang, "Collaborative Hypergraph Networks for Enhanced Disease Risk Assessment," 2024 International Conference on Electronics and Devices, Computational Science (ICEDCS), pp. 416-420, Sept. 2024.

[14] X. Yan, J. Du, L. Wang, Y. Liang, J. Hu, and B. Wang, "The Synergistic Role of Deep Learning and Neural Architecture Search in Advancing Artificial Intelligence," 2024 International Conference on Electronics and Devices, Computational Science (ICEDCS), pp. 452-456, Sept. 2024.

[15] J. Gao, G. Liu, B. Zhu, S. Zhou, H. Zheng, and X. Liao, "Multi-Level Attention and Contrastive Learning for Enhanced Text Classification with an Optimized Transformer," arXiv preprint arXiv:2501.13467, 2025.

[16] Z. Liu, X. Xia, H. Zhang, and Z. Xie, "Analyze the impact of the epidemic on New York taxis by machine learning algorithms and recommendations for optimal prediction algorithms," Proceedings of the 2021 3rd International Conference on Robotics Systems and Automation Engineering, pp. 46-52, May 2021.

[17] Y. Liang, E. Gao, Y. Ma, Q. Zhan, D. Sun, and X. Gu, "Contextual Analysis Using Deep Learning for Sensitive Information Detection," 2024 International Conference on Computers, Information Processing and Advanced Education (CIPAE), pp. 633-637, Aug. 2024.

[18] X. Fei, S. Chai, W. He, L. Dai, R. Xu, and L. Cai, "A Systematic Study on the Privacy Protection Mechanism of Natural Language Processing in Medical Health Records," 2024 IEEE 2nd International Conference on Sensors, Electronics and Computer Engineering (ICSECE), pp. 1819-1824, Aug. 2024.

[19] Z. Liu, M. Wu, B. Peng, Y. Liu, Q. Peng, and C. Zou, "Calibration Learning for Few-shot Novel Product Description," Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 1864-1868, July 2023.

[20] J. Hu, X. Liao, J. Gao, Z. Qi, H. Zheng, and C. Wang, "Optimizing Large Language Models with an Enhanced LoRA Fine-Tuning Algorithm for Efficiency and Robustness in NLP Tasks," arXiv preprint arXiv:2412.18729, 2024.

[21] Y. Wang, Z. Xu, K. Ma, Y. Chen, and J. Liu, "Credit Default Prediction with Machine Learning: A Comparative Study and Interpretability Insights," unpublished.