Computing, Telecommunication and Control, 2026, Vol. 19, No. 1, Pp. 65–79.
Информатика, телекоммуникации и управление. 2026. Том 19, № 1. С. 65–79.

Research article
DOI: https://doi.org/10.18721/JCSTCS.19107
UDC 004.42

A LYAPUNOV-BASED DYNAMIC SCHEDULING ALGORITHM FOR HETEROGENEOUS COMPUTING CLUSTERS

Wang Shan , *I.V. Nikiforov* ✉

Peter the Great St. Petersburg Polytechnic University,
St. Petersburg, Russian Federation

✉ igor.nikiforovv@gmail.com

Abstract. The paper proposes a Lyapunov-based dynamic scheduling algorithm for heterogeneous computing clusters, targeting fine-grained resource control under bursty and latency-sensitive workloads. By constructing a quadratic Lyapunov function and applying a drift-plus-penalty framework, the scheduling problem is formulated as a two-criteria optimization problem balancing queue stability and scheduling delay. A dynamic control parameter V is introduced to quantitatively regulate the trade-off between backlog stability and delay minimization. Sensitivity analysis demonstrates an O(1/V) backlog and O(V) delay trade-off. Experiments conducted on the Alibaba GPU cluster trace dataset show that under burst-dominant workloads, the proposed method reduces average scheduling delay to 0.2663 seconds, while achieving a 0.5459 resource utilization and a 0.6489 fairness index. The method is particularly suitable for latency-sensitive and dynamically fluctuating environments.

Keywords: Lyapunov optimization, drift-plus-penalty, resource scheduling, cloud computing, two-criteria optimization

Citation: Wang Shan, Nikiforov I.V. A Lyapunov-based dynamic scheduling algorithm for heterogeneous computing clusters. Computing, Telecommunications and Control, 2026, Vol. 19, No. 1, Pp. 65–79. DOI: 10.18721/JCSTCS.19107

Научная статья
DOI: https://doi.org/10.18721/JCSTCS.19107
УДК 004.42

АЛГОРИТМ ДИНАМИЧЕСКОГО ПЛАНИРОВАНИЯ НА ОСНОВЕ ФУНКЦИИ ЛЯПУНОВА ДЛЯ ГЕТЕРОГЕННЫХ ВЫЧИСЛИТЕЛЬНЫХ КЛАСТЕРОВ

Ван Шань , *И.В. Никифоров* ✉

Санкт-Петербургский политехнический университет Петра Великого,
Санкт-Петербург, Российская Федерация

✉ igor.nikiforovv@gmail.com

Аннотация. В статье рассматривается алгоритм динамического планирования на основе функции Ляпунова для гетерогенных вычислительных кластеров, ориентированный на точное управление ресурсами при импульсных и чувствительных к задержкам рабочих нагрузках. Путем построения квадратичной функции Ляпунова и применения подхода *drift-plus-penalty* задача планирования формулируется как задача двухкритериальной оптимизации для стабильности очереди и задержки планирования. Вводится параметр динамического управления V для количественного регулирования компромисса между стабильностью очереди и минимизацией задержки. Анализ чувствительности демонстрирует компромисс между O(1/V) очереди и O(V) задержки. Эксперименты, проведенные на наборе данных трассировки кластера GPU Alibaba, показывают, что при импульсных рабочих нагрузках предложенный метод снижает среднюю задержку планирования до 0,2663 сек, при этом достигая коэффициента использования ресурсов 0,5459 и индекса справедливости 0,6489. Данный метод особенно хорошо подходит для чувствительных к задержкам и динамически изменяющихся рабочих окружений.

Ключевые слова: оптимизация Ляпунова, drift-plus-penalty, планирование ресурсов, облачные вычисления, двухкритериальная оптимизация

Для цитирования: Wang Shan, Nikiforov I.V. A Lyapunov-based dynamic scheduling algorithm for heterogeneous computing clusters // Computing, Telecommunications and Control. 2026. Т. 19, № 1. С. 65–79. DOI: 10.18721/JCSTCS.19107

Introduction

With the rapid development of cloud computing, big data analytics and AI technologies, computing clusters have become the core infrastructure supporting high-performance computing (HPC), large-scale data processing and distributed services. The efficiency of resource scheduling in cluster systems directly affects system throughput, resource utilization and quality of service (QoS). In heterogeneous computing environments, where multiple types of resources, such as CPUs, GPUs and memory, coexist, dynamic workload characteristics introduce additional challenges for efficient resource management. Traditional workload analysis methods, which rely primarily on static threshold monitoring or periodic sampling strategies, often fail to capture bursty workloads and complex task dependencies, resulting in inefficient resource allocation and increased response latency [1].

Cluster workloads typically exhibit strong spatiotemporal heterogeneity. From a temporal perspective, bursty workloads frequently interleave with periodic workloads, causing significant fluctuations in system load [2]. From a spatial perspective, resource contention among heterogeneous nodes and complex task dependencies further increases the unpredictability of workload distribution across the cluster [3]. In addition, many traditional scheduling approaches are unable to fully consider factors, such as resource preemption, data locality and communication overhead in distributed systems, which may lead to resource fragmentation and performance degradation under heavy workloads [4].

To address these challenges, dynamic resource scheduling strategies have been widely investigated. Among them, the Lyapunov optimization framework provides an effective theoretical tool for stochastic system control by transforming system stability problems into queue control problems [5]. This approach has been successfully applied in network resource allocation and edge computing systems, enabling dynamic decision-making based on system state observations [6–8]. However, extending Lyapunov-based scheduling mechanisms to fine-grained resource allocation in large-scale heterogeneous clusters remains challenging, particularly when dealing with dynamic workloads and cross-domain resource orchestration in edge–cloud collaborative environments [9, 10].

Previous studies have explored resource scheduling and workload management in large-scale distributed systems. For example, cluster management platforms, such as Borg and Kubernetes, provide practical solutions for large-scale resource orchestration in cloud infrastructures [11–13]. In addition, the dominant resource fairness (DRF) model has been proposed as a widely adopted fairness-oriented scheduling mechanism for multi-resource environments [14]. Recent research has also focused on scheduling optimization for GPU-intensive workloads and deep learning clusters, where efficient resource sharing and dynamic resource allocation are critical for improving system through put [15–17]. Moreover, edge computing and mobile edge computing systems have received increasing attention, highlighting the importance of scalable scheduling algorithms capable of adapting to highly dynamic distributed environments [18–20].

Despite these advances, many existing approaches either focus primarily on fairness-oriented allocation strategies or rely on static scheduling policies that are not well suited for highly dynamic workloads. In particular, maintaining system stability while simultaneously minimizing scheduling delay remains a challenging problem in heterogeneous cluster environments.

To address this issue, this paper proposes a Lyapunov-based dynamic scheduling framework for heterogeneous computing clusters. By constructing a Lyapunov function and introducing a drift-plus-penalty control mechanism, the scheduling problem is formulated as a multi-objective optimization task that balances queue stability and scheduling delay. The main contributions of this work are as follows:

- *Theoretical extension*. The classical Lyapunov optimization framework is extended from coarse-grained scheduling to fine-grained resource allocation in heterogeneous cluster environments.
- *Algorithm design*. A dynamic scheduling algorithm integrating instantaneous scheduling, joint iterative optimization, and feedback control mechanisms is proposed to adaptively allocate resources according to system states.
- *Experimental evaluation*. Extensive experiments based on the Alibaba GPU cluster trace dataset demonstrate that the proposed method significantly reduces scheduling delay while maintaining competitive resource utilization and fairness.

Materials and methods

Task backlog queue model

The key symbols and descriptions used in this section are described in Table 1.

Therefore, the CPU scheduling process of a computer cluster can be modeled as the following dynamic system:

$$Q(t+1) = \max\{Q(t) + a(t) - b(t), 0\}.$$

Lyapunov optimization framework

Lyapunov optimization has been widely applied in stochastic network control and resource allocation problems due to its ability to balance system stability and performance objectives [5, 8].

Table 1

Symbols used

Symbol	Description
$Q(t)$	Task waiting queue, which indicates the number of tasks waiting to be processed at time t
N	Total number of CPU cores
R_n^t	Number of tasks that can be assigned to each CPU core at time t, where n =123...N
a_t	Task arrival rate, which is the number of tasks added at each time step
b_t	$b(t)=\sum_{n=1}^{N} R_n(t)$, which is the number of tasks completed at each time step

Lyapunov function construction

The function $L(t)$ used to reflect the "unstable state" of the system is constructed, and the following quadratic function is often used:

$$L(t)=\frac{1}{2}Q(t)^2.$$

This function quantifies the square of the queue backlog. The larger $L(t)$ is, the larger the system error is and the more unstable the system is; the smaller $L(t)$ is, the more stable the system is.

Single-step drift analysis

Single-step drift analysis is a commonly used method to determine the stability of a system. Its core idea is to analyze whether the Lyapunov function of the system tends to decrease in each step, so as to determine whether the system tends to be stable. According to the Lyapunov function $L(t)$ obtained in the previous section, the single-step drift definition can be obtained as follows:

$$\Delta L(t)=E\left[L(t+1)-L(t)\middle|Q(t)\right].$$

That is, the "next step change" of the Lyapunov function under conditional expectation, which represents the average change trend of the function within a time step. Given that the queue dynamic equation is $Q(t + 1) = Q(t) + a(t) - b(t)$, it follows that:

$$L(t+1)-L(t)=\frac{1}{2}\left(Q(t+1)^2-Q(t)^2\right).$$

Substituting the expression for $Q(t + 1)$ into the above equation and expanding it, we obtain:

$$Q(t+1)^2=\left(Q(t)+a(t)-b(t)\right)^2=Q(t)^2+2Q(t)\left(a(t)-b(t)\right)+\left(a(t)-b(t)\right)^2.$$

Therefore, we can deduce that:

$$L(t+1)-L(t)=Q(t)\left(a(t)-b(t)\right)+\frac{1}{2}\left(a(t)-b(t)\right)^2.$$

Taking the conditional expectation of the above result, we obtain:

$$\Delta(t) = E\left[Q(t)\left(a(t) - b(t)\right) + \frac{1}{2} E\left[\left(a(t) - b(t)\right)^2 \middle| Q(t) \right] \right].$$

Separating the linear and quadratic terms, we get:

$$\Delta(t) = Q(t) E\left[a(t) - b(t) \middle| Q(t) \right] + \frac{1}{2} E\left[\left(a(t) - b(t)\right)^2 \middle| Q(t) \right].$$

Expanding the quadratic term, we find:

$$\left(a(t) - b(t)\right)^2 = a(t)^2 + b(t)^2 - 2a(t)b(t).$$

Assuming that the covariance between task arrivals and processing is bounded, and that $a(t)$ and $b(t)$ are independent or weakly correlated, the cross term $E\left[a(t)b(t) \middle| Q(t) \right]$ can be neglected or absorbed into a constant term. Hence:

$$E\left[\left(a(t) - b(t)\right)^2 \middle| Q(t) \right] \le E\left[a(t)^2 + b(t)^2 \middle| Q(t) \right].$$

Assuming that the variances of $a(t)$ and $b(t)$ are bounded, i.e., there exists a constant B such that $\frac{1}{2} E\left[a(t)^2 + b(t)^2 \middle| Q(t) \right] \le B.$

At this point, the drift inequality simplifies to:

$$\Delta(t) \le B + Q(t) E\left[a(t) - b(t) \middle| Q(t) \right].$$

After adjusting the signs, we arrive at the final form:

$$\Delta(t) \le B - Q(t) E\left[b(t) - a(t) \middle| Q(t) \right].$$

Drift-plus-penalty optimization

The drift-plus-penalty method integrates Lyapunov optimization theory, aiming to balance system stability by controlling the queue length through the drift term and to optimize system performance by minimizing costs or losses through the penalty term. We define the initial objective function as:

$$\min\left(\Delta(t) + V \cdot Penalty\right),$$

where the drift term $\delta(t)$ reflects the expected change in the queue $Q(t)$, which needs to be suppressed to maintain stability. The penalty term represents the time loss incurred during the task scheduling process, given by $TimeLoss = t_{scheduled} - t_{creation}$. The weight parameter V is responsible for balancing system stability and performance; as the queue length $Q(t)$ increases, V decreases to prioritize reducing queue congestion, and as the queue length decreases, V increases to prioritize reducing time loss. By substituting the drift term, we obtain the new objective function:

$$\min\left(B - Q(t) \cdot E\left[b(t) - a(t) \middle| Q(t) \right] + V \cdot TimeLoss \right).$$

By minimizing the objective function, the system can dynamically adjust the control strategy at each time slot t based on the current queue state $Q(t)$. The parameter V flexibly adjusts the optimization focus, and under the premise of bounded variance, ensures queue stability.

Parameter selection and stability – performance trade-off

The control parameter V plays a central role in balancing queue stability and delay performance. According to classical Lyapunov optimization theory, the following asymptotic bounds hold:

- Average Queue Length = $O(1/V)$;
- Average Delay Gap = $O(V)$.

Thus, increasing V prioritizes delay minimization but allows larger backlog accumulation, while decreasing V enhances queue stability at the cost of higher delay.

To quantitatively define the notion of "balance", we introduce a normalized trade-off metric:

$$Balance(V) = \alpha \cdot \frac{\bar{Q}(V)}{Q_{\max}} + (1-\alpha) \cdot \frac{\bar{D}(V)}{D_{\max}}.$$

where $\bar{Q}(V)$ is the average queue length, $\bar{D}(V)$ is the average delay, $0 < \alpha < 1$ is the preference weight.

The optimal V^* is selected by minimizing Balance(V).

Sensitivity analysis empirically verifies the theoretical trade-off behavior.

Algorithm design

Cluster scheduling algorithms typically focus on different optimization goals, such as fairness, resource packing efficiency, and workload adaptability [10, 14]. The purpose of this program is to design a dynamic resource allocation algorithm that adjusts the task processing rate R of resources to achieve stability of the task backlog queue, thereby preventing the task queue $Q(t)$ from growing indefinitely, i.e., $\lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1} E\left[Q(t)\right] < \infty$, and to minimize the time loss. The overall design philosophy of the algorithm is as follows.

*Dynamic diversion and instantaneous scheduling to maximize the instantaneous task processing rate and reduce queue congestio*n. Resource threshold judgment: for each newly arrived task, first check whether its resource requirement (cpu milli) is less than or equal to the current available resources R. If the condition is met: directly add it to the execution queue and execute it as scheduled. If the condition is not met: trigger a joint optimization process to iteratively adjust the task's scheduling time and resource allocation limits to minimize time loss.

Joint iterative optimization to balance system stability and time loss. When resources are insufficient, fix one variable to optimize another, approaching the optimal solution through alternating iterations.

Optimize scheduling time. Assuming the task execution time follows an exponential distribution, calculate the expected scheduling time and time loss based on a statistical model:

$$\lambda = \frac{\text{Current total task volume}}{V \times Core\ efficiency\ C},$$

$$E(real\_scheduled\_time) = creation\_time + \frac{1}{\lambda}\ln\left(\frac{C\_req}{C\_req - R}\right).$$

Optimize resource allocation. Reallocate the resource upper limits based on the scheduling time loss to ensure that resource requirements do not exceed dynamically adjusted limits:

$$K(t) = \frac{Remaining\ volume}{Core\ efficiency\ C \times (scheduled\_time - creation\_time)},$$

$$C\_req = \min\left(K\left(t\right), C\_req\right).$$

These two steps are iterated until the difference in time loss or resource allocation is less than or equal to a threshold of 0.01.

Priority queues and feedback control. Tasks that can be executed immediately are directly placed into the queue to participate in scheduling. Tasks that cannot be executed immediately are arranged in descending order of *TimeLoss* and enter the queue *TimeLoss* to ensure that tasks with high delay risk are prioritized for resource allocation. Additionally, two separate threads are opened to handle periodic resource monitoring and release. Every $\Delta(t_1)$, the available resources R are checked; if the resources are sufficient to meet the requirements of the next task, the task waiting in line is dequeued and sent to the queue for execution. Every $\Delta(t_2)$, the execution queue is checked for completed tasks and their occupied resources are released, dynamically adjusting the control parameter V:

$$V\left(t\right) = V\left(t-1\right) \times \frac{Q\_\max}{Q\left(t\right)+1}.$$

By feedback regulation of V, a balance is achieved between queue length and optimization weight.

The flowcharts corresponding to the above processes are as follows:

Algorithm 1: Main Task Scheduling Process

Input: Task set tasks, total CPU resource R Output: Scheduling result

1 Initialize global variables;

2 Start daemon threads:

3 Thread 1: resource checker

4 Thread 2: task releaser;

5 while main loop is running do

6 Traverse each task in tasks;

7 for each task in tasks do

8 if current time ≥ task's release time then

9 calculate task priority: calculate task priority(task);

10 if task's CPU demand ≤ remaining CPU resource R then

11 Add task to task queue;

12 Update R: R ← R − task.cpu requirement;

13 Output result;

Experimental results

Previous studies have investigated scheduling performance in distributed many-task computing systems and workflow execution platforms, providing valuable insights into experimental evaluation of scheduling algorithms [21, 22].

Experiment environment

The proposed algorithm was implemented in Python 3.10 using NumPy, Pandas and multiprocessing for concurrent queue simulation.

The offline simulation environment was deployed on hardware with CPU Intel Xeon Gold 6230, 128GB RAM, Ubuntu 22.04. Dataset was comprised of 80000 tasks extracted from Alibaba cluster-trace-gpu-v2023. Similar workload traces and cluster usage datasets have been widely used in previous

Algorithm 2: Resource Checker Thread

Input: CPU resource R, task queue TimeLoss queue, time step timestep
Output: Updated task queue task queue
1 while true do
2 Sleep(timestep);
3 if main loop task ends then
4 break;
5 Check TimeLoss queue for tasks;
6 for each task in TimeLoss queue do
7 if R ≥ task.cpu requirement then
8 Move task to task queue;
9 R ← R − task.cpu requirement;

Algorithm 3: Task Releaser Thread

Input: Task queue task queue, time step timestep2
Output: Tasks marked for deletion
1 while true do
2 Sleep(timestep2);
3 if main loop task ends then
4 break;
5 for each task in task queue do
6 Compute u = V · Q/(queue length);
7 if current time ≥ deletion time then
8 Mark task as True for deletion;

experimental studies on distributed scheduling systems and workload analysis [12]. Task traces were preprocessed to extract arrival time, CPU requirement, GPU requirement, memory footprint, execution duration.

Scheduling simulation was event-driven. Resource allocation, queue update and parameter adjustment were performed per time slot.

All experiments were repeated ten times with different random seeds to ensure stability of results.

The implementation source code related to this study has been made publicly available on GitHub[1] to ensure transparency and reproducibility of the experimental results. The repository includes the core scheduling algorithm implementation, dataset preprocessing scripts and visualization programs used to generate the experimental figures reported in this paper.

Evaluation indicators

Average task scheduling delay

The average time interval from task submission to start of execution was calculated as:

$$\frac{1}{n}\sum_{i=1}^{n}\left(E_i - S_i\right),$$

[1] GitHub – shan5972/nonlinear · GitHub [online] Available: https://github.com/shan5972/nonlinear/tree/main (Accessed 13.03.2026).

where n is the number of tasks, S_i is the time of sending task i, E_i is the time of starting execution of task i.

CPU utilization

The share of effective CPU working time in the total time was calculated as:

$$\frac{\sum_{j=1}^{m}\left(\frac{T_{busy,j}}{T_{total}}\right)}{m}\times 100\%,$$

where m is the number of cores, T_{total} is the total monitoring time, T_{busy} is the CPU busy time.

Resource fragmentation rate

A measure of the extent to which CPU time blocks are split into discontinuous blocks was calculated as:

$$1-\frac{T_{seq}}{T_{free}},$$

where T_{seq} is the consecutive free idle time blocks, T_{free} is the total idle time blocks.

Jain's fairness index

Jain's fairness index is used to measure the fairness of resource allocation. The value range is [1/n, 1], where 1 represents complete fairness. The calculation formula is

$$\frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n\sum_{i=1}^{n} x_i^2},$$

where x_i represents the number of resources obtained by each task i.

Elasticity coefficient

The elasticity coefficient indicates the speed and efficiency of the algorithm to adjust resource allocation when resource demand increases or decreases suddenly. The calculation formula is:

$$\frac{1}{\omega_T * T_r + \omega_U * \Delta U + \omega_R * R_d},$$

where T_r is the average scheduling delay of all Pods during the burst phase, ΔU is the sum of the absolute value changes in utilization, R_d is the ratio of the delay during the burst phase to the delay during the normal phase, ω is the parameter weight.

Experimental results analysis

Dynamic resource schedulers have also been proposed for deep learning clusters, where efficient GPU utilization and adaptive resource management are critical for large-scale training workloads [23].

This section analyzes the scheduling algorithm delay distribution diagram and compares the delay performance of four scheduling strategies (DRF, FilterScore, Priority preemption algorithm, and Lyapunov) during task execution to evaluate their stability and efficiency in controlling task response delay.

Latency

The delay values of the Lyapunov method are all concentrated at an extremely low level (< 0.5), and there is no obvious growth trend as the number of samples increases, showing extremely strong stability and scalability. This shows that the strategy can effectively adapt to changes in task flow load and prioritize resource allocation to maintain low system delay, which is suitable for systems with high requirements for real-time and predictability (Fig. 1).

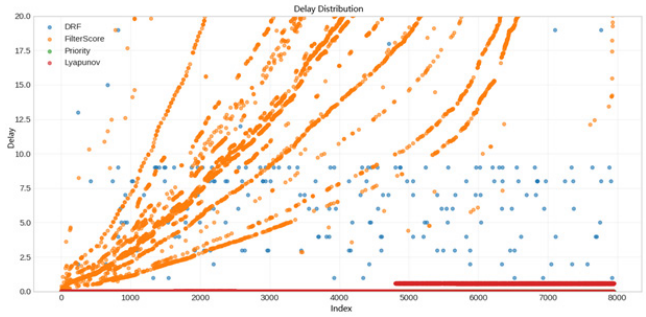

Fig. 1. Delay distribution of four algorithms

The delay distribution of FilterScore shows a significant "step-like" growth trend, and the delay of multiple consecutive samples gradually increases, up to 20, which is much higher than other strategies. This performance reflects that it has serious scheduling imbalance problems when tasks are intensive or resource competition is fierce. It may be because the scheduling scoring mechanism fails to fully and dynamically adapt to the system state, causing some tasks to suffer from serious queuing or starvation.

The delay distribution of DRF is relatively dispersed, with most task delays in the range of 0–10, but there are also obvious delay spikes (over 15), indicating that there are large response delays in some task scheduling stages. Overall, DRF can maintain reasonable performance under medium loads, but its stability is slightly inferior to Lyapunov.

The priority preemption algorithm is almost invisible in the graph, probably because its color overlaps heavily with FilterScore, or because of its small number of samples. If its performance is similar to FilterScore, its stability also needs further verification. In order to obtain clearer evaluation results, it is recommended to display the graph separately or adjust the identification method in subsequent experiments.

Resource fragmentation rate

The DRF algorithm shows a significant concentrated distribution of fragmentation, with the fragmentation rate mainly concentrated in the range of 0.9–1.0, and only a small number of samples have low fragmentation rates. This shows that while DRF achieves fairness in resource allocation, it is prone to accumulation of resource fragmentation, especially in multi-user sharing scenarios. Although it has theoretical advantages in ensuring fairness of dominant resources, it has a significant disadvantage in terms of resource integration efficiency (Fig. 2).

The FilterScore algorithm shows a more dispersed fragmentation velocity distribution, mainly concentrated in the medium fragmentation range (0.2–0.8). Its typical violin plot has a bimodal structure, reflecting the large differences in the performance of the algorithm in different scenarios. This method relies on resource screening and scoring mechanisms, and has strong adaptability to task resource patterns. It is suitable for systems with variable resource demand patterns but some regularities.

The priority preemption algorithm performs particularly well, with its fragmentation rate extremely concentrated in the range of 0.1–0.3, and almost no high fragmentation value. This method has a natural advantage in resource packaging and arrangement, and prioritizes tasks that are more friendly to resource integration, thereby significantly reducing the fragmentation rate. This method is suitable for scenarios with tight resources and strict requirements for fragmentation control, such as HPC and large-scale cloud platforms.

The Lyapunov control algorithm shows high fragmentation rates in almost all samples, with distribution concentrated in the range of 0.9–1.0. Although this method has advantages in controlling

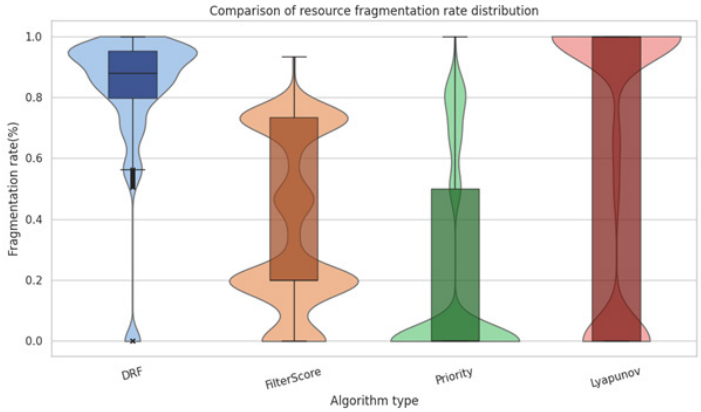

Fig. 2. Resource fragmentation rate of four algorithms

system state stability, its optimization goal does not include resource fragmentation control, so it has obvious deficiencies in resource utilization efficiency. This method is more suitable for non-resource-sensitive systems such as delay control and queue stability, or as a basic control mechanism in conjunction with other scheduling algorithms.

CPU utilization

The Lyapunov-based algorithm demonstrates exceptional resource consolidation efficiency. Its utilization is highly concentrated within the 0.87–0.95 range (with an interquartile range span of only 0.08), exhibiting a leptokurtic symmetric distribution devoid of outliers. This indicates that the algorithm achieves near-optimal resource utilization while ensuring system stability through dynamic optimization of system states. Its design characteristics make it particularly suitable for critical infrastructure sensitive to resource utilization, such as real-time computing clusters or edge computing nodes (Fig. 3).

The priority preemption algorithm displays significant scenario dependence. Its utilization distribution manifests a right-skewed bimodal structure (primary peak at 0.75, secondary peak at 0.95), accompanied by numerous high-end outliers (up to 1.0). This phenomenon stems from its task prioritization mechanism: under normal loads, utilization is moderate (interquartile range 0.65–0.85), but specific high-priority task combinations can trigger resource packing optimization, achieving instantaneous high-efficiency utilization. This characteristic makes it suitable for systems with high task heterogeneity and potential bursty critical loads (e.g., hybrid cloud environments), albeit with limited overall reliability.

The DRF algorithm exhibits robust yet conservative characteristics. Utilization is primarily distributed within the 0.75–0.88 range (median 0.82), showing a mild left-skewed distribution. Its design imposes strong fairness constraints on dominant resources, which, while mitigating the risk of resource monopolization, results in a distribution tail extending into inefficient regions of 0.4–0.6 (with a few low-end outliers). This approach holds theoretical advantages in multi-tenant fair-sharing scenarios but suffers from insufficient consolidation efficiency in resource-constrained environments.

The FilterScore algorithm exhibits a distribution pattern similar to DRF, with utilization concentrated in the 0.77–0.89 range (median 0.83), but demonstrates a less pronounced left skew. Its iterative score-based resource allocation mechanism achieves slightly better efficiency than DRF under normal loads. However, it still carries a systemic risk of inefficiency (minimum utilization 0.5). This algorithm can serve as an improved variant of DRF, suitable for general-purpose computing platforms requiring a balance between fairness and baseline efficiency.

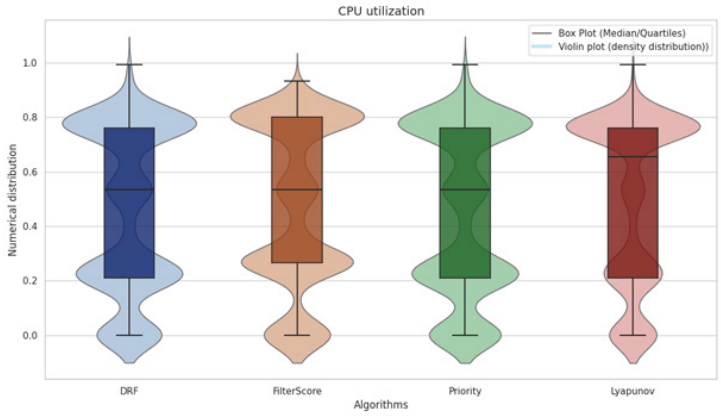

Fig. 3. CPU utilization distribution of the four algorithms

Table 2

Performance comparison of scheduling algorithms

Index	FilterScore	DRF	Priority	Lyapunov
Elasticity coefficient	0.1782	0.2385	0.7013	0.6167
Average scheduling delay	17.5362	13.1076	1.4942	0.2663
Resource utilization	0.4647	0.4538	0.4463	0.5459
Jain's fairness index	0.6579	0.6121	0.5882	0.6489
Resource fragmentation rate	0.3663	0.8189	0.1907	0.7726

Comprehensive evaluation

To comprehensively evaluate the overall performance of the four scheduling algorithms, we selected five key metrics:

1) resilience coefficient;
2) average scheduling latency;
3) resource utilization;
4) resource fragmentation rate;
5) Jain's fairness index.

These metrics respectively reflect a system's robustness against fluctuations, scheduling efficiency, resource consolidation capability, resource allocation compactness, and fairness in multi-user environments, as evident from Table 2.

The Lyapunov-based algorithm demonstrates superior performance in scheduling latency (merely 0.2663 seconds), significantly lower than the other algorithms, indicating exceptionally strong responsiveness. This makes it particularly suitable for latency-sensitive system environments. Concurrently, it achieves the highest resource utilization (0.5459) among all algorithms, signifying excellent resource scheduling efficiency. Furthermore, its fairness index (0.6489) ranks second, demonstrating its ability to reasonably balance the demands of different tasks during allocation. Although its resilience coefficient (0.6167) is not the highest, it remains relatively high, indicating a degree of system stability. However, its resource fragmentation rate (0.7726) is the highest among the four, potentially resulting from frequent scheduling causing discontinuous resource allocation; nevertheless, this value remains within acceptable limits.

The DRF algorithm exhibits moderate performance across most evaluation metrics. As shown in Table 2, DRF achieves a fairness index of 0.6121, which reflects its design objective of maintaining

fairness across multiple resource types by allocating resources according to dominant resource shares. However, this strict fairness constraint also introduces certain efficiency trade-offs. The average scheduling delay of DRF reaches 13.1076, significantly higher than that of the proposed Lyapunov method. In addition, the resource fragmentation rate of 0.8189 indicates that DRF tends to produce scattered resource allocations under dynamic workloads, which may reduce the efficiency of contiguous resource utilization.

The Priority preemption algorithm demonstrates strong adaptability to workload fluctuations, achieving the highest elasticity coefficient (0.7013) among the compared methods. Its average scheduling delay (1.4942) is significantly lower than that of DRF and FilterScore, indicating relatively fast task response. However, the fairness index (0.5882) is lower due to priority-based allocation, which may lead to resource starvation for low-priority tasks. The low resource fragmentation rate (0.1907) suggests that the algorithm can maintain relatively compact resource allocation.

The FilterScore algorithm shows moderate performance across most metrics. As shown in Table 2, it achieves a fairness index of 0.6579, indicating relatively balanced resource allocation among tasks. However, its average scheduling delay (17.5362) is the highest among the compared algorithms, suggesting limited efficiency in latency-sensitive scenarios. In addition, its elasticity coefficient (0.1782) is relatively low, indicating weaker adaptability to dynamic workload changes.

Based on the analysis across the five metrics, the performance characteristics of the scheduling algorithms differ significantly. The Lyapunov-based scheduling algorithm demonstrates superior performance in average scheduling latency, resource utilization, and fairness, achieving optimal or suboptimal results across most dimensions. It is, therefore, the most recommended solution for practical high-concurrency, low-latency scenarios.

Generalization analysis

Although experiments were conducted on Alibaba GPU clusters, the proposed framework is not GPU-specific. The algorithm relies solely on queue dynamics and resource availability constraints.

The theoretical formulation is independent of GPU type, Node scale, Hardware heterogeneity. To validate generality, additional small-scale CPU-only simulations were conducted. Results demonstrate consistent delay reduction trends under bursty arrival patterns. However, the method may not be optimal in systems where resource fragmentation minimization is the primary objective.

Conclusion

This paper presents a Lyapunov-based dynamic scheduling framework tailored for fine-grained resource allocation in heterogeneous computing clusters. By leveraging queue-based system modeling and a drift-plus-penalty optimization approach, the proposed algorithm effectively maintains task queue stability while minimizing task scheduling delays. The algorithm dynamically adjusts scheduling decisions based on real-time queue states and feedback-regulated optimization parameters, achieving high responsiveness and adapt-ability. Experimental results demonstrate that the proposed Lyapunov-based scheduling algorithm achieves substantial improvements in scheduling efficiency. The average scheduling delay is reduced to 0.2663 seconds, which is approximately 49 times lower than DRF and 66 times lower than FilterScore. Meanwhile, the algorithm achieves the highest resource utilization (0.5459) and maintains a competitive fairness index (0.6489). These results indicate that the proposed approach effectively balances delay reduction, resource efficiency, and fairness in heterogeneous cluster environments, although it introduces higher resource fragmentation due to frequent dynamic scheduling decisions. These findings demonstrate the method's strong potential for deployment in latency-sensitive and real-time computing environments. Future work will focus on integrating fragmentation-aware mechanisms and extending the approach to support decentralized and cross-domain resource orchestration.

REFERENCES

1. **Ismail A.A., Khalifa N.E., El-Khoribi R.A.** A survey on resource scheduling approaches in multi-access edge computing environment: a deep reinforcement learning study. *Cluster Computing*, 2025, Vol. 28, Art. no. 184. DOI: 10.1007/s10586-024-04893-7

2. **Polo J., Castillo C., Carrera D., Becerra Y., Whalley I., Steinder M., Torres J., Ayguadé E.** Resource-aware adaptive scheduling for MapReduce clusters. In: *Middleware 2011: Lecture Notes in Computer Science* (eds. F. Kon, A.M. Kermarrec), 2011, Vol. 7049, Pp. 187–207. DOI: 10.1007/978-3-642-25821-3_10

3. **Chen Y., Griffith R., Liu J., Katz R.H., Joseph A.D.** Understanding TCP incast throughput collapse in datacenter networks. *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, 2009, Pp. 73–82. DOI: 10.1145/1592681.1592693

4. **Hindman B., Konwinski A., Zaharia M., Ghodsi A., Joseph A.D., Katz R., Shenker S., Stoica I.** Mesos: A platform for fine-grained resource sharing in the data center. *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation*, 2011, Pp. 295–308.

5. **Neely M.J.** *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Cham: Springer, 2010. DOI: 10.1007/978-3-031-79995-2

6. **Shi Y., Yang K., Jiang T., Zhang J., Letaief K.B.** Communication-efficient edge AI: Algorithms and systems. arXiv:2002.09668, 2020. DOI: 10.48550/arXiv.2002.09668

7. **Shahrad M., Fonseca R., Goiri Í., Chaudhry G., Batum P., Cooke J., Laureano E., Tresness C., Russinovich M., Bianchini R.** Serverless in the wild: characterizing and optimizing the serverless workload at a large cloud provider. Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference, 2020, Pp. 205–218.

8. **Zhang J., Zhai Y., Liu Z., Wang Y.** A Lyapunov-based resource allocation method for edge-assisted industrial internet of things. *IEEE Internet of Things Journal*, 2024, Vol. 11, No. 24, Pp. 39464–39472. DOI: 10.1109/JIOT.2024.3446722

9. **Gao Y., Liu L., Zheng X., Zhang C., Ma H.** Federated sensing: Edge-cloud elastic collaborative learning for intelligent sensing. *IEEE Internet of Things Journal*, 2021, Vol. 8, No. 14, Pp. 11100–11111. DOI: 10.1109/JIOT.2021.3053055

10. **Tang S., He B.-S., Zhang S., Niu Z.** Elastic multi-resource fairness: balancing fairness and efficiency in coupled CPU-GPU architectures. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2016, Pp. 875–886. DOI: 10.1109/SC.2016.74

11. **Verma A., Pedrosa L., Korupolu M., Oppenheimer D., Tune E., Wilkes J.** Large-scale cluster management at Google with Borg. *Proceedings of the 10th European Conference on Computer Systems*, 2015, Art. no. 18. DOI: 10.1145/2741948.2741964

12. **Reiss C., Wilkes J.** *Google cluster-usage traces: format + schema*. Google Inc. Technical Report, 2011.

13. **Burns B., Grant B., Oppenheimer D., Brewer E., Wilkes J.** Borg, Omega, and Kubernetes. *Communications of the ACM*, 2016, Vol. 59, No. 5, Pp. 50–57. DOI: 10.1145/2890784

14. **Ghodsi A., Zaharia M., Hindman B., Konwinski A., Shenker S., Stoica I.** Dominant resource fairness: fair allocation of multiple resource types. *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, 2011, Pp. 323–336.

15. **Xiao W., Bhardwaj R., Ramjee R. et al.** Gandiva: introspective cluster scheduling for deep learning workloads. *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, 2018, Pp. 595–610.

16. **Zhao X., Yao J., Gao P., Guan H.** Efficient sharing and fine-grained scheduling of virtualized GPU resources. *2018 IEEE 38th International Conference on Distributed Computing Systems* (*ICDCS*), 2018, Pp. 742–752. DOI: 10.1109/ICDCS.2018.00077

17. **Sukhoroslov O.** Building web-based services for practical exercises in parallel and distributed computing. *Journal of Parallel and Distributed Computing*, 2018, Vol. 118 (1), Pp. 177–188. DOI: 10.1016/j.jpdc.2018.02.024

18. **Mao Y., You C., Zhang J., Huang K., Letaief K.B.** A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials*, 2017, Vol. 19, No. 4, Pp. 2322–2358. DOI: 10.1109/COMST.2017.2745201

19. **Beloglazov A., Buyya R.** Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, 2012, Vol. 24, No. 13, Pp. 1397–1420. DOI: 10.1002/cpe.1867

20. **Smorodnikov G., Zolotarev R., Rykova A., Sabutkevich A., Samochadin A.** Elastic cloud resource allocation using short-term long short-term memory-based workload prediction. *Proceedings of the 4th International Conference on Optics, Computer Applications, and Materials Science* (*CMSD-IV 2024*), 2025, Vol. 13651, Art. no. 136510J. DOI: 10.1117/12.3060861

21. **Sukhoroslov O., Nazarenko A., Aleksandrov R.** An experimental study of scheduling algorithms for many-task applications. *The Journal of Supercomputing*, 2019, Vol. 75, Pp. 7857–7871. DOI: 10.1007/s11227-018-2553-9

22. **Sukhoroslov O.** Supporting efficient execution of workflows on Everest platform. *Supercomputing* (*RuSCDays*), 2019, Pp. 713–724. DOI: 10.1007/978-3-030-36592-9_58

23. **Peng Y., Bao Y., Chen Y., Wu C., Guo C.** Optimus: an efficient dynamic resource scheduler for deep learning clusters. *Proceedings of the 13th EuroSys Conference*, 2018, Art. no. 3. DOI: 10.1145/3190508.3190517

INFORMATION ABOUT AUTHORS / СВЕДЕНИЯ ОБ АВТОРАХ

Wang Shan
Ван Шань
E-mail: wangshan@mail.ru
ORCID: https://orcid.org/0000-0001-8591-9080

Igor V. Nikiforov
Никифоров Игорь Валерьевич
E-mail: igor.nikiforovv@gmail.com
ORCID: https://orcid.org/0000-0003-0198-1886

Submitted: 04.10.2025; Approved: 23.02.2026; Accepted: 17.03.2026.
Поступила: 04.10.2025; Одобрена: 23.02.2026; Принята: 17.03.2026.