Automation and Machine Learning (2025)
Clausius Scientific Press, Canada

DOI: 10.23977/autml.2025.060113
ISSN 2516-5003 Vol. 6 Num. 1

A Multi-Level Feedback Queue Optimization Method Based on Reinforcement Learning and Dynamic Time Slice Scheduling

Zhong Wenxuan

School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou, 310018, Zhejiang, China

Keywords: Reinforcement Learning; Dynamic Time Slice Scheduling; Operating System; Multi-Level Feedback Queue

Abstract: This invention pertains to the field of reinforcement learning and discloses a multi-level feedback queue optimization method [2] based on reinforcement learning and dynamic time slot scheduling. It involves separately obtaining process feature data, system load data, user experience data, and hardware-related data. The acquired data is preprocessed to ensure accuracy and reliability. Based on the preprocessed data, the data sets for process features, system load, user experience, and hardware-related are formed through numbering. Using these numbered datasets, the process feature index, system load rate, user experience index, and hardware load rate are calculated. A comprehensive evaluation of queue priority values is conducted, and multi-level feedback queue optimization is implemented based on these evaluated queue priority values. This approach considers multiple dimensions of scheduling methods, enhancing the quality and accuracy of scheduling decisions, avoiding certain processes from occupying too many resources and affecting the operation of other processes, making multi-level feedback queue optimization more equitable.

1. Background Introduction

The Multi-levelFeedbackQueue (MFQ) scheduling algorithm is an efficient method used for task scheduling in operating systems, aiming to optimize CPU usage and system response time. In computer systems, the allocation of processor resources is one of the key issues in operating system design. Traditional scheduling algorithms such as First-Come-First-Served (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin (RR) each have their advantages and disadvantages, failing to fully meet the demands of modern complex computing environments. With the rapid increase in servers and tasks, and the growing complexity of resource scheduling methods, there is a need for a scheduling strategy that can adapt to various types of processes and optimize system performance. The MultilevelFeedbackQueue (MFQ) scheduling algorithm has emerged to address these needs. It combines the benefits of Round Robin and Priority Scheduling, dynamically adjusting the priority of tasks in different queues to achieve efficient management and utilization of system resources.

The existing multi-level feedback queue scheduling algorithms primarily focus on the allocation and scheduling of CPU resources, with relatively less consideration given to other system resources such as memory and I/O devices. However, in actual systems, the demand for resources like memory by processes also impacts system performance. Currently, multi-level feedback queue scheduling does not adequately incorporate the usage of memory and other resources into its decision-making process, which may result in some processes failing to run properly, affecting overall resource utilization. Additionally, it lacks a certain degree of fairness.

In response to the shortcomings of existing technologies, this paper presents a multi-level feedback queue optimization method based on reinforcement learning and dynamic time slot scheduling. This approach comprehensively considers multiple dimensions in scheduling, which helps improve the quality and accuracy of scheduling decisions, reduce resource waste and conflicts, and achieve more balanced resource allocation. It prevents certain processes from consuming too many resources and affecting the operation of other processes, ensuring that high-priority or high-user experience index processes receive better service. Consequently, it enhances overall service quality and makes multi-level feedback queue optimization more equitable.

2. Introduce reinforcement learning strategy in multi-level feedback queue

This invention addresses the shortcomings of existing technologies and proposes a multi-level feedback queue scheduling method that combines reinforcement learning with dynamic time slot adjustment. This method optimizes scheduling decisions by introducing a deep Q network (DQN) and identifies whether the system's process tasks are primarily CPU-intensive or I/O-intensive to dynamically adjust the time slot sizes of each queue. Through simulation comparisons, it can significantly reduce frequent context switches, adapt well to task loads, and enhance system throughput, thereby significantly improving system efficiency. This method can help optimize process scheduling algorithms in actual operating system development, enabling the system to achieve higher throughput, which is of great significance for the development of process scheduling in operating systems.

First, we need to establish a basic multi-level feedback mechanism, allowing the system to set multiple priority queues for tasks and initialize different time slice lengths for each queue: high-priority queues have shorter time slices; low-priority queues have longer time slices, with subsequent queue time slice lengths remaining constant; when a new task arrives, it is initially placed in the highest priority queue; if the task is not completed within the current queue's time slice, it will be moved to the next lower priority queue; high-priority tasks preempt the execution resources of low-priority tasks; if tasks in low-priority queues remain un-scheduled for an extended period, the priority of all processes can be increased through an aging (Aging) mechanism to prevent "hunger" issues.

We set the observation state space as the saqueue length and the action space as the selection of the execution queue.

We use the DQN model to select the best queue according to the queue length εstate. The training adopts the experience recovery and-greedy strategy, and the Q value update formula is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max\left(Q(s',a')\right) - Q(s,a)\right] \tag{1}$$

Among them, is the $sar\ \gamma\alpha$current state, is the action, is the immediate reward, is the discount factor, and is the learning rate.

This formula is the core of the learning algorithm$Q(S,a)QQ(s,a)QQ(s,a)$, which is used to update the state-action pair and the value of the state-action pair. The value represents the long-term expected return of taking action a under state s. Through continuous iterative update, it gradually approaches the optimal value, so as to guide the scheduler to choose the best action.

The training optimization DQN mechanism is defined as follows: rewards are allocated according to the execution results of tasks, with +1 for completed tasks, -0.1 for unfinished tasks, and-1 for empty queue selection. The total reward is calculated as follows:

$$R = \sum\left(reward_{complete} + reward_{imcomplete} + reward_{empty}\right) \tag{2}$$

The R value obtained by this formula is used to train and optimize the DQN strategy.

This total reward R is used as the goal of the deep Q network QR(DQN) to drive the learning algorithm to optimize the decision. DQN learns to select the best queue by maximizing the expected value, so as to improve the system throughput.

Next, we need to set the context switching penalty: when an unfinished task is downgraded to the next queue or returned to the original queue, the switching penalty time is increased, $C = k \cdot switch_{count} switch_{count}$which is denoted as: where k is the word switching penalty value and is the number of times the context is switched.

The formula explains that after a task is taken from the selected queue, the execution time is the smaller value of the remaining time and the time slice. Completing the task rewards +1; failing to complete it results in a queue demotion and a penalty of-0.1, with an empty queue receiving a penalty of-1. The DQN dynamically decides which queue to execute based on the current queue status (the number of tasks in each queue), rather than strictly following the priority order of queue numbers. This is the first optimization point discussed in this paper.

3. Introduce dynamic time slice adjustment strategy in multi-level feedback queue

Due to the traditional multi-machine feedback queue scheduling where the time slice length [5] for each queue is fixed, in actual high-load scenarios, segment tasks may experience increased waiting times due to overly long time slices (even the shortest ones), while long tasks may be frequently interrupted due to overly short time slices (even the longest time slice in multi-level feedback queues). Therefore, we differ from the traditional principle of fixing the time slice lengths for each queue after initialization. Instead, we dynamically adjust the time slice lengths based on the type of background task and the required time slice length during actual execution, thereby improving scheduling efficiency!

After every 10 rounds of training in DQN, we adjust the time slice according to the proportion of $exhaustion_{radio}$process task time slice exhaustion:

If so, it is determined to be CPU-intensive and the time slice is doubled:$exhaustion_{radio} < low_{level_{exhaustion}}$

$$T' = [2t_0, 2t_1 \dots, 2t_{n-1}] \tag{3}$$

This means that the vast majority of task processes are CPU-intensive, so the time slice length of each queue is doubled from the original to reduce frequent context switching.

If so, it is determined to be I/O intensive and the time slice is doubled:$exhaustion_{radio} < low_{level_{exhaustion}}$

$$T' = [\max(1, t_0/2), \max(1, t_1/2), \dots, max(1, t_{n-1}\}/2)] \tag{4}$$

This means that most of the task processes are I/O intensive, so the time slice length of each queue is halved from the original.

4. Specific implementation methods

As mentioned above, we provide a multi-level feedback queue scheduling method combining reinforcement learning [1,2] and dynamic time slot adjustment, which can reduce the overhead of redundant context switching compared with traditional multi-level feedback queue, improve throughput, and adapt to high load in actual situations. [3,4]

As shown in Figure 1, the overall model structure of this invention includes the (running queue structure), (priority array (scheduling entity) and (task structure) required by the traditional multi-level feedback $struc_{rq}struct_{rt_{prio_{array}}}, struct\ sched\_rt\_entitystruct\ task\_struct$ queue, as well as the DQN module and dynamic time slice adjustment module unique to this patent, in order to improve the load and throughput.

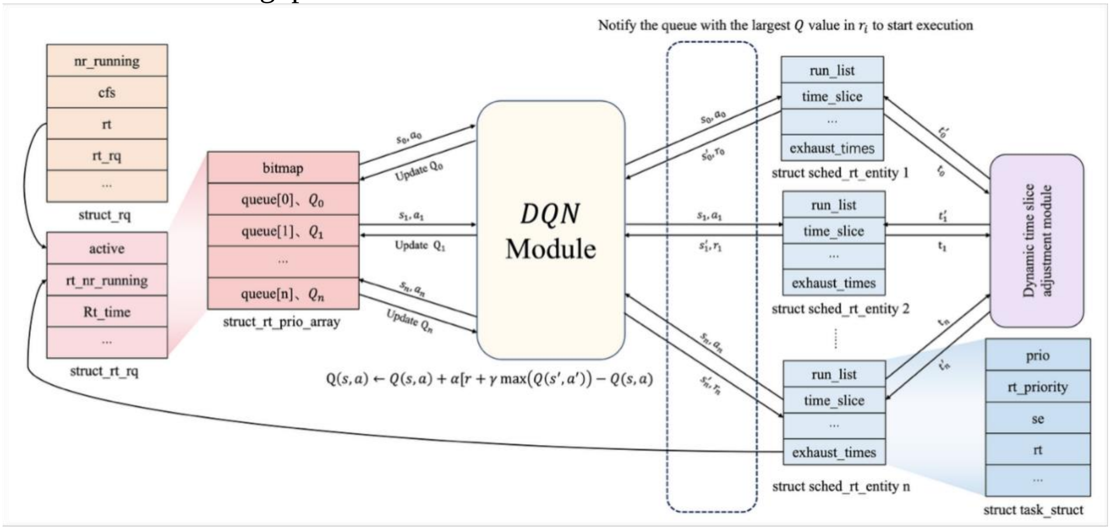
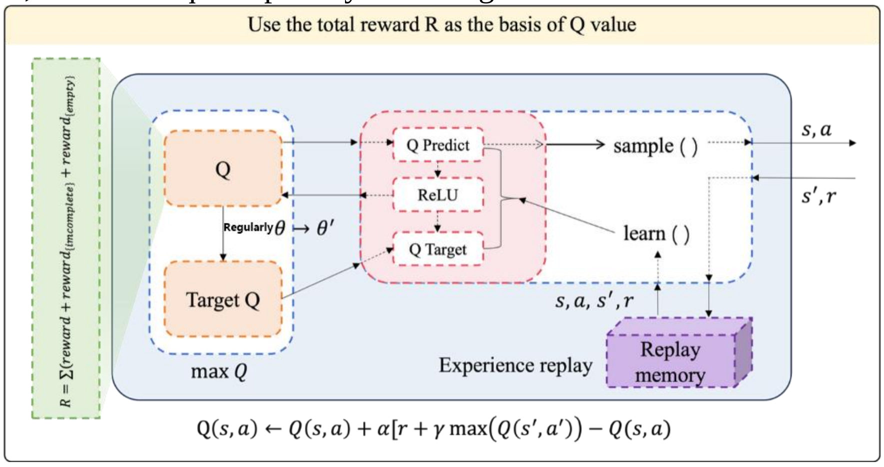


Figure 1: Overall structure of the model of the invention

First, construct the basic multi-level feedback queue mechanism as shown in Figure 1. The system sets multiple priority queues for tasks, each with different time slice lengths; when a new task arrives, it is initially placed in the highest priority queue; if the task is not completed within the current time slice, it will be moved to the next lower priority queue; high-priority tasks preempt the execution resources of low-priority tasks; if a task in a low-priority queue remains un-scheduled for an extended period, its priority can be increased through the aging (Aging) mechanism to prevent starvation issues.

On the basis of the original structure, new member variables are added $struct\ priority\ arrayQ_i$to calculate the queue score, which is used as the basis for subsequent dynamic priority adjustment of the queue.

The deep network Q(DQN) is implemented to dynamically select the execution queue according to the queue state, so that the queue priority is no longer fixed.



Use the total reward R as the basis of Q value
$R = \sum(reward + reward_{(imcomplete)} + reward_{(empty)})$
Q
Regularly $\theta \rightarrow \theta'$
Target Q
max Q
Q Predict
ReLU
Q Target
sample ( )
learn ( )
s, a
s', r
s, a, s', r
Experience replay
Replay memory
$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max(Q(s',a')) - Q(s,a)$

Figure 2: Internal structure of DQN

Specifically, as shown in Figure 2, the internal structure of DQN, we use the defined total reward QRfunction (Formula 2) as the value basis, and the larger the value, the higher the score.

Among them, DQN is used to update the Q value Q(s, a) of the state-action pair (S, a). The Q value represents the long-term expected return of taking action a under the state s. Through continuous iterative update, Q(s, a) gradually approaches the optimal value, so as to guide the scheduler to choose the best action.

We used ReLU as the loss function, and Figure 3 shows the internal parameter network of DQN: both the input layer and output layer consist of queue Q values. The middle part includes two hidden layers with a total of 64 parameters (it should be noted that in practice, the parameter values of these internal hidden layers may need to be adjusted according to the complexity of the actual task and production requirements to achieve optimal results).

When the process scheduling begins, every 10 rounds check if all processes in the queue have exhausted their time slices. If the exhaustion ratio exceeds $radiohigh_levelradiolow_level$the maximum threshold, it is determined that the majority of process tasks are CPU-intensive. To reduce context switching and increase system throughput, the time slices for each queue are doubled. Conversely, if the exhaustion ratio is below the minimum threshold, it is determined that the majority of process tasks are I/O-intensive. To reduce overhead and increase system throughput, the time slices for each queue are reduced.

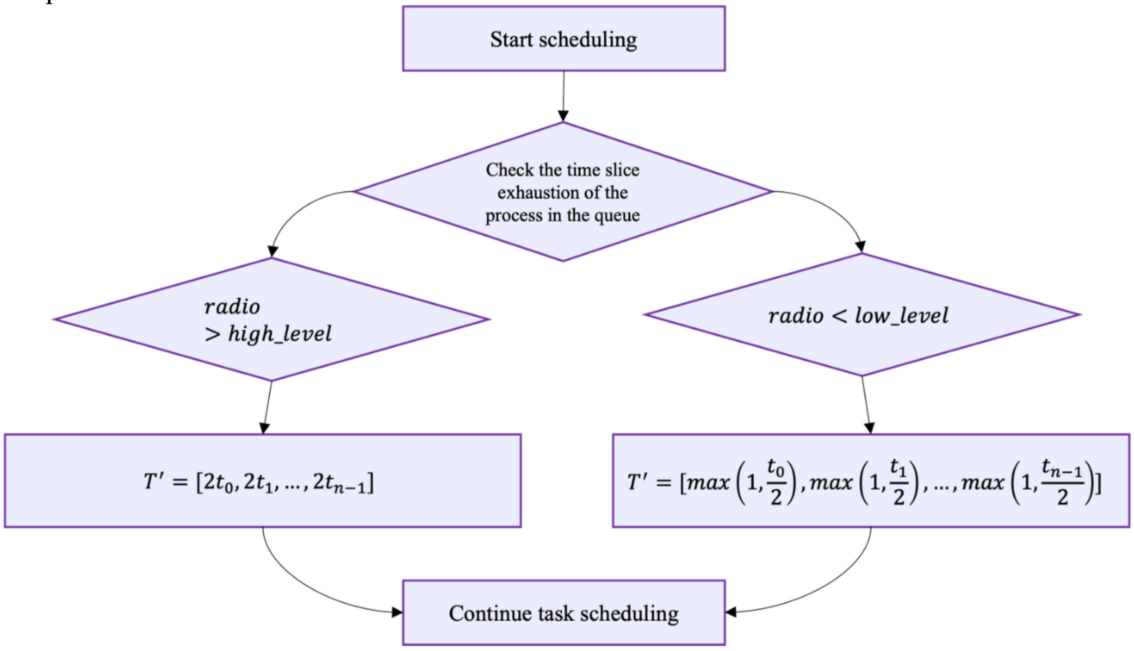

Start scheduling
Check the time slice exhaustion of the process in the queue
$radio > high\_level$
$radio < low\_level$
$T' = [2t_0, 2t_1, \dots, 2t_{n-1}]$
$T' = [max\left(1, \frac{t_0}{2}\right), max\left(1, \frac{t_1}{2}\right), \dots, max\left(1, \frac{t_{n-1}}{2}\right)]$
Continue task scheduling

Figure 3: Dynamic time slice adjustment flow chart

5. Numerical experimental verification

As a preferred option, the evaluation method of the multi-level feedback queue scheduling method combining reinforcement learning and dynamic time slot adjustment is:

$$\text{Turnaround Time}_{\text{Average}} = \frac{\sum(\text{Time}_{\text{end}} - \text{Time}_{\text{arrival}})}{\text{n}}$$

$$\text{Wait Time}_{\text{Average}} = \frac{\sum(\text{Turnaround Time} - \text{Execution Time})}{\text{n}}$$

$$\text{Handling Capacity} = \frac{\text{n}}{\text{Total Time}}$$

$$\text{Utilization Rate}_{\text{CPU}} = \frac{\sum \text{Execution Time}}{\text{Total Time}}$$

In this experimental example, our experimental results are based on the simulation data to compare the performance of the invention "multi-level feedback queue scheduling method combining reinforcement learning and dynamic time slice adjustment" with the traditional multi-level feedback queue (MFQS) algorithm and the benchmark algorithm (single time slice scheduling).

In order to reduce the randomness and more conform to the real situation, we set 100 tasks, in which the arrival $\lambda = 5$time of tasks follows poisson distribution (); the execution time of tasks follows uniform distribution (10ms to 100ms);

The performance of the traditional multi-level feedback queue (MFQS), the benchmark algorithm (single time slice) and the invention (DQN+ dynamic adjustment) was compared in the experiment, and the results are shown in the following table 1:

Table 1: Comparison of scheduling algorithm performance

Algorithm	Throughput (tasks/second)	Average waiting time (ms)	Total completion time (ms)	Context switching time	Total rewards
Tradition MFQS	85	40	382	60	10.0
Baseline (single time slice 4)	80	45	393	71	8.9
This invention (DQN + dynamic adjustment)	100	30	340	18	14.2

Throughput: This invention improves by 18% (100 vs 85), outperforming traditional MFQS and benchmark algorithms; Throughput: This invention improves by 18% (100 vs 85), outperforming traditional MFQS and benchmark algorithms; Average Wait Time: This invention reduces by 25% (30ms vs 40ms), making task responses faster; Total Completion Time: This invention decreases by 11% (340ms vs 382ms), enhancing overall efficiency; Context Switches: This invention reduces by 70% (18 vs 60), significantly lowering system overhead. Total Rewards: This invention increases by 42% (14.2 vs 10.0), with more optimized scheduling strategies.

6. Summary

This paper proposes a multi-level feedback queue scheduling method that combines reinforcement learning and dynamic time slot adjustment. In practical applications, this method can effectively address the issue of fixed time slots for each queue in traditional multi-level feedback queue scheduling algorithms, which often fails to meet complex load requirements. By adopting the DQN strategy, we allocate the most suitable queue for each process task, rather than uniformly placing them in the highest priority queue first. We can then dynamically adjust the time slot lengths of each queue based on the type and proportion of system tasks (whether the task is CPU-intensive or I/O-intensive) to improve throughput and reduce unnecessary context switching.

References

[1] Li Z, Ierapetritou M. Process scheduling under uncertainty: Review and challenges[J]. Computers & Chemical Engineering, 2008, 32(4-5): 715-727.

[2] Harki N A. Multi-level feedback queue scheduling technique[J]. Cihan University-Erbil Scientific Journal, 2024, 8(2): 36-42.

[3] Kilminster S, Zukas M, Quinton N, et al. Preparedness is not enough: understanding transitions as critically intensive learning periods[J]. Medical education, 2011, 45(10): 1006-1015.
[4] Suresh V, Chaudhuri D. Dynamic scheduling—a survey of research[J]. International journal of production economics, 1993, 32(1): 53-63.
[5] Branke J, Mattfeld D C. Anticipation and flexibility in dynamic scheduling[J]. International Journal of Production Research, 2005, 43(15): 3103-3129.*