Multi-Queue Scheduler for the CD-MOJ Platform

Lucas G. Caldas[1], **Bruno C. Ribas**[2]

[1]Faculdade do Gama (FGA) – Universidade de Brasília (UnB)
Brasília – DF – Brazil

lucasgcaldas01@gmail.com, bruno.ribas@unb.br

Abstract. *This paper presents the implementation of a multi-queue scheduler in the CD-MOJ system, which is used in programming courses and competitions at UnB FGA. The goal is to improve submission correction efficiency by prioritizing tasks based on their importance. Results show a reduction in response time from* 12.33 *to* 9.86 *seconds, representing a* 20.03% *improvement. The work highlights the importance of efficient scheduling to optimize resource allocation and user experience in online judge platforms.*

Resumo. *Este artigo apresenta a implementação de um escalonador de múltiplas filas no sistema CD-MOJ, utilizado em disciplinas de programação e competições na UnB FGA. O objetivo é melhorar a eficiência na correção de submissões, priorizando-as conforme sua importância. Os resultados mostram uma redução no tempo de resposta de* $12,33$ *para* $9,86$ *segundos, representando uma melhoria de* $20,03\%$*. O trabalho ressalta a importância de um escalonamento eficiente para otimizar o uso de recursos e a experiência do usuário em plataformas de juízes online.*

1. Introduction

The Contest-Driven Meta Online Judge [CD-MOJ], developed at the University of Brasília (UnB), is an online judge [Wasik et al. 2018] used for algorithm courses. It provides a vast array of problems for exams and training activities but faces challenges with high submission volumes and limited computational resources. The varied difficulty levels and problem complexities result in some problems having long time limits while others are nearly instantaneous per test instance. The current system processes submissions semi-sequentially, using a First-Come, First-Served (FCFS) algorithm until all machines are occupied. This method can cause delays when handling large volumes or complex problems. This paper proposes a multi-queue scheduling algorithm to address these issues, prioritizing submissions based on importance and enhancing resource allocation and response times.

This paper is structured as follows: Section 2 provides a brief theoretical background, Section 3 details our proposed solution, Section 4 evaluates our initial findings, and Section 5 offers our preliminary conclusions and future work.

2. Theoretical Background

To understand the proposed solution, it's essential to review the basics of scheduling in operating systems. Scheduling is the method by which tasks are assigned to CPU resources for execution. Common algorithms include:

- **Round Robin**: Each process gets a fixed time quantum. If a task exceeds this time, it moves to the queue's end [Tanenbaum and Bos 2016].
- **Priority Scheduling**: Tasks are executed based on assigned priorities, ensuring high-priority tasks are processed first. This can lead to starvation for low-priority tasks [Tanenbaum and Bos 2016].
- **Multi-level Queue Scheduling**: Combines various scheduling algorithms and sorts tasks into multiple queues based on their characteristics (e.g., interactive or batch). Each queue follows different scheduling rules, and tasks can be dynamically shifted between queues [Silberschatz and Galvin 2015].

In the context of CD-MOJ, the most suitable approach is a hybrid solution, where submissions are divided into multiple queues based on their importance, allowing efficient prioritization and fair resource distribution.

3. Proposed Solution

The proposed solution introduces a multi-queue scheduler into the CD-MOJ system. Each submission is classified into one of four priority queues: super priority for critical tasks, exam queue for submissions during exams, private list for internal exercises, and public list for general submissions. The scheduler dynamically assigns submissions to available machines based on priority. A starvation prevention mechanism ensures that low-priority submissions are promoted to higher-priority queues if they have been waiting for more than five minutes.

The solution includes modifications to existing scripts and introduces a new central scheduler script. This script continuously monitors submission queues and machine availability, assigning tasks based on priority and machine capacity. An aging mechanism is implemented to promote long-waiting submissions, preventing indefinite delays. As shown in Listing 1, the scheduler operates in two phases: first, jobs that require specific servers are assigned; second, jobs that can run on any server are handled.

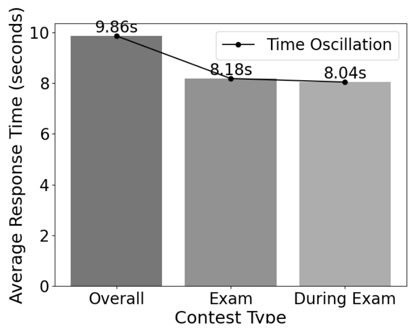
Listing 1. Scheduler pseudo-code

```
function run_scheduler() {
    initialize available_servers;
    initialize selected_jobs;

    // Phase 1: Assign jobs needing specific servers
    for each job_queue in priority_queues; do
        for each job in job_queue; do
            if no available servers then break;
            if job requires specific server then
                assign to available specific server;
                lock and move job to scheduled list and update metadata;
                remove server from available_servers;
            end if;
        end for;
    end for;

    // Phase 2: Assign jobs that can run on any server
    for each job_queue in priority_queues; do
        for each job in job_queue; do
            if no available servers then break;
            if job can run on any server then
                assign to first available server;
                lock and move job to scheduled list and update metadata;
                remove server from available_servers;
            end if;
        end for;
    end for;
}
```

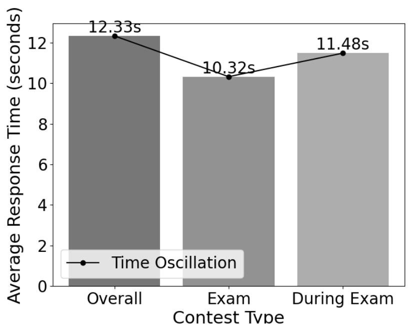
4. Experiments

The proposed system was tested on real-world data from the CD-MOJ platform, with over $60,000$ submissions analyzed. Before the implementation of the scheduler, as shown in Figure 1(a), the average response time for a submission was 12.33 seconds. After implementing the scheduler, as shown in Figure 1(b), the average response time was reduced to 9.86 seconds, representing a 20.03% improvement.

During high-demand periods, such as exams, the scheduler was able to reduce response times significantly by prioritizing critical submissions. For example, during an exam, the average response time decreased from 10.32 seconds to 8.18 seconds, a substantial improvement in providing timely feedback to students.



(a) Fig. 1 - Avg. Response Time by Contest Type: No Scheduler



(b) Fig. 2 - Avg. Response Time by Contest Type: With Scheduler

5. Conclusion

The multi-queue scheduler in the CD-MOJ system effectively manages high volumes of submissions and optimizes resource allocation. The decreased average response time highlights the system's ability to prioritize critical submissions while preventing lower-priority task starvation. Future improvements include exploring task preemption, parallelizing test case correction for enhanced performance, and removing the two-phase mechanism to distribute jobs across available machines in a single iteration.

References

CD-MOJ. Cd-moj documentation. `https://cd-moj.github.io/cd-moj.docs/`. Accessed on 20 de novembro de 2023.

Silberschatz, A. and Galvin, Peter Baer e Gagne, G. (2015). *Fundamentos de Sistemas Operacionais*. LTC, Rio de Janeiro, 9ª edição.

Tanenbaum, A. S. and Bos, H. (2016). *Sistemas Operacionais Modernos*. Pearson Education do Brasil, São Paulo, 4ª edição. Tradução Jorge Ritter; revisão técnica Raphael Y. de Camargo.

Wasik, S., Maj, P., Orzechowski, P., Wiszniewski, B., and Zielinski, K. (2018). A survey on online judge systems and their applications. *ACM Computing Surveys*, 51(1).