Universidade de Brasília – UnB
Campus Gama – FGA
Engenharia de Software

Escalonamento de Múltiplas Filas no CD-MOJ

Lucas Gomes Caldas

Orientador: Prof. Dr. Bruno César Ribas

UnB – Universidade de Brasília

FGA – Faculdade Gama

Engenharia de Software

Escalonamento de Múltiplas Filas no CD-MOJ

Lucas Gomes Caldas

Orientador: Prof. Dr. Bruno César Ribas

Trabalho de Conclusão de Curso
Engenharia de Software

Brasília/DF, Julho de 2024

UnB – Universidade de Brasília

FGA – Faculdade Gama

Engenharia de Software

Escalonamento de Múltiplas Filas no CD-MOJ

Lucas Gomes Caldas

Trabalho de Conclusão de Curso submetido à Faculdade UnB Gama da Universidade de Brasília, como parte dos requisitos necessários para a obtenção do grau de bacharel em Engenharia de Software

Aprovada por:

Prof. Dr. Bruno César Ribas

(Orientador)

Prof. Dr. John Lenon Cardoso Gardenghi

Prof. Dr. Daniel Saad Nogueira Nunes

Brasília/DF, Julho de 2024

Ficha Catalográfica

Caldas, L. G.
Escalonamento de Múltiplas Filas no CD-MOJ,
[Distrito Federal], 2024.
29p., 210 × 297 mm (FGA/UnB Gama, Bacharelado em Engenharia de Software, 2024).
Trabalho de Conclusão de Curso, Faculdade UnB Gama, Engenharia de Software

1. Juíz Online	2. Sistema Operacional
3. Engenharia de Software	4. A definir
I. FGA UnB/UnB.	II. Título (série)

Referência

Caldas, L. G. (2024). Escalonamento de Múltiplas Filas no CD-MOJ. Trabalho de Conclusão de Curso, Engenharia de Software, Faculdade UnB Gama, Universidade de Brasília, Brasília, DF, 29p.

Cessão de Direitos

---

lucasgcaldas01@gmail.com

Brasília, DF – Brasil

Resumo

Este trabalho aborda a implementação de um escalonador para o sistema CD-MOJ (*Contest-Driven Meta Online Judge*), uma plataforma desenvolvida para facilitar competições de programação na Universidade de Brasília - Faculdade do Gama (UnB FGA). O objetivo principal é melhorar a eficiência do sistema de correção de submissões, gerenciando de forma mais eficaz a alocação de recursos computacionais. A implementação de um escalonador visa atribuir diferentes níveis de prioridade às submissões, baseando-se na sua importância, para reduzir o tempo médio de correção e proporcionar um *feedback* mais rápido aos usuários.

O escalonador proposto utiliza múltiplas filas de prioridade, onde as submissões são categorizadas e processadas conforme sua relevância. As filas são organizadas de forma a evitar a inanição de submissões de baixa prioridade, promovendo-as após um período específico na fila. O modelo desenvolvido inclui *scripts* modificados e novos, que gerenciam a distribuição das submissões e a comunicação entre os servidores de correção.

A implementação foi testada e validada em cenários reais, mostrando uma redução significativa no tempo médio de resposta, que passou de 12,33 segundos para 9,86 segundos, representando uma melhoria de 20,03%. Durante provas específicas, a eficiência do sistema também foi comprovada com uma redução considerável no tempo de correção.

A análise dos resultados, destacou a eficácia do escalonador em otimizar a utilização das máquinas disponíveis e reduzir o tempo de espera das submissões. O trabalho demonstra a importância de um sistema de escalonamento eficiente para gerenciar submissões em plataformas de juiz *online*, melhorando significativamente a experiência dos usuários e a alocação de recursos computacionais.

Abstract

This work presents the implementation of a scheduler for the CD-MOJ (Contest-Driven Meta Online Judge) system, a platform developed to facilitate programming competitions at the University of Brasília - Faculty of Gama (UnB FGA). The main objective is to improve the efficiency of the submission correction system by more effectively managing the allocation of computational resources. The implementation of a scheduler aims to assign different priority levels to submissions based on their importance, in order to reduce the average correction time and provide faster feedback to users.

The proposed scheduler uses multiple priority queues, where submissions are categorized and processed according to their relevance. The queues are organized to avoid starvation of low-priority submissions, promoting them after a specific period in the queue. The developed model includes modified and new scripts that manage the distribution of submissions and the communication between correction servers.

The implementation was tested and validated in real scenarios, showing a significant reduction in average response time, which decreased from 12.33 seconds to 9.86 seconds, representing an improvement of 20.03%. During specific exams, the system's efficiency was also proven with a considerable reduction in correction time.

The analysis of the results highlighted the effectiveness of the scheduler in optimizing the utilization of available machines and reducing the waiting time for submissions. The work demonstrates the importance of an efficient scheduling system to manage submissions in online judge platforms, significantly improving the user experience and the allocation of computational resources.

Sumário

Lista de Tabelas

Lista de Figuras

1 Introdução

O CD-MOJ (*Contest-Driven Meta Online Judge*) [1], uma plataforma de juiz *online* mantida pelo Prof. Dr. Bruno César Ribas e voltada para competições de programação na UnB FGA - Universidade de Brasília - Faculdade do Gama, foi desenvolvida para simplificar a elaboração de competições e correções de soluções, oferecendo uma ferramenta de correção imediada aos alunos e uma base de exercícios para os professores. Atualmente, o sistema do CD-MOJ enfrenta desafios significativos no que diz respeito à gestão e alocação eficiente de recursos computacionais para a correção das submissões. Isso ocorre devido à possibilidade de a única fila de execução ficar bloqueada, dependendo do tempo limite estabelecido para cada problema, da quantidade de submissões pendentes e da disponibilidade de máquinas para realizar as correções necessárias.

O CD-MOJ dispõe de um conjunto de 10 máquinas distribuídas em grupos, cada uma com especificações diferentes para atender às variadas demandas de correção. Esta plataforma tem proporcionado um bom serviço à comunidade da FGA, satisfazendo disciplinas como APC (Algoritmos e Programação de Computadores), FAC (Fundamentos de Arquitetura de Computadores), FSO (Fundamentos de Sistemas Operacionais), Estrutura de Dados I e Estrutura de Dados II, assim como disciplinas optativas e de pós-graduação. Ao longo do ano de 2024, conforme estatísticas extraídas do sistema, o número de submissões já ultrapassou a marca de 60 mil. Esse alto volume de submissões evidencia a importância e a forte demanda por essa plataforma, demonstrando como ela se tornou eficiente ao lidar com a quantidade expressiva de atividades submetidas pelos alunos.

Embora a plataforma atenda às necessidades da comunidade de maneira geral, ainda existem oportunidades para aprimoramentos. A ausência de mecanismos eficientes para dividir os casos de teste entre as máquinas disponíveis e priorizar o uso dessas máquinas e submissões no CD-MOJ acaba levando a tempos prolongados na execução dos casos de teste. Isso prejudica o processo de avaliação, já que cada máquina é responsável por executar todos os casos de teste de cada submissão de maneira individual e sequencial.

A ineficiência no tratamento e identificação das submissões ocorre devido à falta de um sistema de priorização eficaz para as listas ou provas na plataforma. Essa lacuna na

gestão de prioridades compromete significativamente a eficiência do sistema, dificultando o atendimento a prazos críticos para as correções. A necessidade de priorização se torna clara ao considerar um exemplo onde uma submissão leva 6 horas para ser corrigida. Se todas as máquinas estiverem processando submissões desse tipo, qualquer nova submissão de outra disciplina terá de esperar 6 horas antes que a correção possa começar. Isso impede um *feedback* rápido para o usuário da outra disciplina, afetando negativamente a experiência do usuário e a capacidade do sistema de fornecer respostas ágeis e efetivas.

1.1 Objetivos

1.1.1 Objetivos Gerais

Tendo isso em vista, o objetivo deste trabalho é implementar um escalonador projetado para atribuir diferentes níveis de prioridade a cada submissão, permitindo que elas sejam processadas não apenas pela ordem de chegada, mas também com base em sua importância, reduzindo assim o tempo médio geral de correção. Essa abordagem visa atender de maneira mais eficaz às demandas e prioridades específicas, garantindo que as submissões prioritárias recebam a atenção necessária no tempo adequado.

1.1.2 Objetivos Específicos

- Analisar o funcionamento atual do módulo de correção do CD-MOJ.
- Identificar e categorizar as diferentes submissões com base em critérios de prioridade.
- Desenvolver e implementar um escalonador com múltiplas filas de prioridades.
- Testar e validar o desempenho do escalonador em cenários reais de uso.
- Comparar os tempos de resposta e eficiência do sistema com e sem o escalonador.
- Reduzir o tempo de espera na fila para as submissões em geral.
- Melhorar a experiência do usuário ao proporcionar um feedback mais rápido e eficiente.

2 Fundamentação Teórica

2.1 Escalonamento em Sistemas Operacionais

Quando um computador é multiprogramado, vários processos ou *threads* frequentemente competem pela CPU (*Central Processing Unit*) ao mesmo tempo, exigindo uma escolha sobre qual processo será executado em seguida. O escalonador do sistema operacional faz essa escolha com base em um algoritmo de escalonamento [2].

Escalonamento em sistemas operacionais refere-se ao processo de gerenciamento e distribuição equitativa dos recursos do sistema, principalmente a CPU, entre vários processos que competem por esses recursos. O objetivo principal do escalonamento é otimizar o desempenho do sistema, maximizando a eficiência do uso da CPU e minimizando o tempo de resposta para os processos. O escalonamento também pode envolver outros recursos além da CPU, como a alocação de recursos de I/O (*input/output*) e a memória, garantindo a eficiência global do sistema operacional [2].

Em diferentes ambientes de computação, variados algoritmos de escalonamento são necessários, principalmente devido às metas distintas em diferentes áreas de aplicação e tipos de sistemas operacionais. Três ambientes notáveis são:

1. Ambiente em Lote:
   (a) Usado para tarefas empresariais periódicas.
   (b) Algoritmos não preemptivos ou com longos períodos para cada processo.
   (c) Chaveamentos de processo minimizados para otimizar o desempenho.
2. Ambiente Interativo:
   (a) Preempção essencial para evitar que um processo monopolize a CPU.
   (b) Riscos de processos no estado `bloqueado` prejudicando o serviço para outros.
   (c) Inclui servidores remotos atendendo a múltiplos usuários necessitando de respostas rápidas.

3. Ambiente de Tempo Real:

   (a) Preempção nem sempre necessária, pois processos frequentemente bloqueiam rapidamente.

   (b) Execução de programas dedicados a um propósito específico.

   (c) Diferencia-se dos sistemas interativos que podem executar programas arbitrários, até mesmo mal-intencionados.

Em sistemas de lote, maximizar a vazão (número de tarefas por hora), minimizar o tempo de retorno e manter a CPU ocupada são prioridades. No entanto, maximizar a vazão nem sempre reduz o tempo de retorno, especialmente quando diferentes tipos de tarefas estão envolvidas. Em sistemas interativos, a ênfase está na minimização do tempo de resposta, atendendo rapidamente às solicitações dos usuários e atendendo às suas expectativas de tempo. Em sistemas de tempo real, o cumprimento dos prazos é crucial para evitar perda de dados, e a previsibilidade é vital, principalmente em sistemas multimídia [2].

Com isso, o algoritmo de escalonamento deve atender as várias metas, dependendo do ambiente operacional. A justiça é crucial, garantindo tratamento equitativo para processos comparáveis, enquanto a aplicação das políticas do sistema é essencial para cumprir regras estabelecidas. Manter o sistema ocupado é uma meta geral, otimizando a utilização da CPU e dos dispositivos de I/O. Ou seja, para o contexto do CD-MOJ, é interessante abordar algoritmos de escalonamento em sistemas interativos, devido as suas características e propostas da plataforma.

2.2 Algoritmo Round Robin

O algoritmo de escalonamento *round-robin*, também conhecido como circular, é um dos métodos mais antigos e amplamente utilizados na gestão de processos em um sistema operacional. Nele, cada processo recebe um intervalo de tempo chamado de *quantum*, geralmente variando entre 10 a 100 milissegundos [3], durante o qual é permitido executar na CPU. Quando o *quantum* de um processo termina, ocorre uma preempção, e a CPU passa para outro processo disponível na fila de execução [2].

Se um processo termina ou é bloqueado antes do término do *quantum*, a CPU realiza a troca de processo no momento do bloqueio ou término. O algoritmo pode ser implementado da seguinte forma: o escalonador mantém uma fila de processos executáveis, a qual é tratada como uma fila FIFO (*First-In*, *First-Out*) [3] e, quando um processo usa

todo o seu *quantum*, ele é movido para o final da lista para dar oportunidade aos outros processos [2].

Um ponto crucial nesse algoritmo é a duração do *quantum*. O tempo necessário para trocar de um processo para outro, chamado de "chaveamento de contexto", demanda recursos da CPU, como salvar e carregar registros, atualizar tabelas e listas, além de manipular a memória *cache*. Se o *quantum* for muito curto em relação ao tempo de chaveamento de contexto, uma parte significativa do tempo da CPU será desperdiçada em atividades administrativas, reduzindo sua eficiência [2].

Por outro lado, se o *quantum* for muito longo em comparação com o tempo médio de uso da CPU pelos processos, a preempção ocorrerá com pouca frequência. Isso pode ser considerado benéfico, já que os processos tendem a terminar ou ser bloqueados antes do término do *quantum*, minimizando as trocas de processo. No entanto, um *quantum* longo pode resultar em respostas lentas para solicitações interativas curtas, especialmente se houver uma grande variação nas demandas de CPU dos processos na fila [2].

Em resumo, a escolha do tamanho do *quantum* de tempo é crucial para o desempenho desse algoritmo. Deve ser grande o suficiente para evitar trocas de contexto excessivas, mas não tão grande a ponto de comprometer a eficiência do escalonamento.

2.3 Algoritmo por Prioridades

O escalonamento por prioridades é um método que pressupõe que todos os processos têm a mesma importância, atribuindo a cada um deles uma prioridade específica. Esse modelo de escalonamento baseia-se na ideia de que os processos com maior prioridade devem ser executados primeiro. Mesmo em sistemas com apenas um único usuário, certos processos podem necessitar de prioridades mais altas em relação aos demais, seja por sua relevância ou por requisitos específicos [2].

Há diversas maneiras de lidar com as prioridades no escalonamento. Uma delas é a diminuição da prioridade do processo em execução a cada interrupção de relógio, o que evita que processos com prioridades elevadas monopolizem a CPU indefinidamente. Outro método é estabelecer um *quantum* máximo de tempo para cada processo executar antes de ceder a vez para o próximo com prioridade mais alta. As prioridades podem ser estáticas, definidas manualmente, ou dinâmicas, ajustadas pelo sistema para alcançar determinados objetivos ou otimizar o serviço [2].

Existem várias maneiras de atribuir prioridades aos processos. Por exemplo, em

ambientes Linux, em que os processos com prioridade negativa são os mais importantes, aqueles com prioridade zero têm uma prioridade normal e os processos com prioridade maior que zero são considerados de baixa prioridade.

No entanto, a desvantagem dos algoritmos de prioridades é o risco de bloqueio indefinido ou inanição, onde processos de baixa prioridade podem ficar esperando indefinidamente para utilizar a CPU. Em sistemas altamente carregados, processos de prioridade mais alta podem monopolizar a CPU, deixando os de baixa prioridade em espera prolongada [3].

Uma solução para esse problema é o conceito de "envelhecimento", que envolve o aumento gradual da prioridade dos processos que estão esperando na fila por um longo período. Dessa forma, mesmo processos de baixa prioridade eventualmente receberão prioridade mais alta e, consequentemente, a oportunidade de serem executados. Esse método ajuda a evitar bloqueios indefinidos e garante que todos os processos tenham a chance de serem executados, independentemente de sua prioridade inicial [3].

2.4 Fila multinível

Tanto no escalonamento por prioridade quanto no escalonamento circular (*round-robin*), todos os processos podem ser colocados em uma única fila, e o escalonador seleciona o processo com a maior prioridade para execução. Na prática, é mais fácil ter filas separadas para cada prioridade distinta, e o escalonamento por prioridade simplesmente agenda o processo na fila de maior prioridade. Essa abordagem, conhecida como fila multinível, também funciona bem quando o escalonamento por prioridade é combinado com o *round-robin*: se houver vários processos na fila de maior prioridade, eles são executados em ordem *round-robin*. Na forma mais generalizada dessa abordagem, uma prioridade é atribuída estaticamente a cada processo, e o processo permanece na mesma fila durante toda a sua execução.

Um algoritmo de escalonamento de fila multinível também pode ser usado para dividir processos em várias filas separadas com base no tipo de processo. Por exemplo, uma divisão comum é feita entre processos interativos e processos em lote. Esses dois tipos de processos têm diferentes requisitos de tempo de resposta e, portanto, podem ter diferentes necessidades de escalonamento. Além disso, processos interativos podem ter prioridade sobre processos em lote. Filas separadas podem ser usadas para processos interativos e em lote, e cada fila pode ter seu próprio algoritmo de escalonamento. A fila interativa pode ser escalonada por um algoritmo *round-robin*, por exemplo, enquanto a fila em lote é escalonada por um algoritmo FCFS (*First-Come*, *First-Served*).

Além disso, deve haver escalonamento entre as filas, que é comumente implementado como escalonamento preemptivo de prioridade fixa. Por exemplo, a fila de processos em tempo real pode ter prioridade absoluta sobre a fila interativa.Um exemplo de algoritmo de escalonamento de fila multinível com quatro filas, em ordem de prioridade, seria:

1. Processos em tempo real
2. Processos do sistema
3. Processos interativos
4. Processos em lote

Cada fila tem prioridade absoluta sobre as filas de menor prioridade. Nenhum processo na fila de processos em lote poderia ser executado a menos que as filas de processos em tempo real, processos do sistema e processos interativos estivessem todas vazias. Se um processo interativo entrasse na fila de pronto enquanto um processo em lote estivesse sendo executado, o processo em lote seria preemptado.

Outra possibilidade é dividir o tempo entre as filas. Aqui, cada fila recebe uma certa porção do tempo da CPU, que pode ser escalonada entre seus vários processos. Por exemplo, na divisão de filas interativos ou em lote, a fila interativa pode receber 80% do tempo da CPU para escalonamento *round-robin* entre seus processos, enquanto a fila do em lote recebe 20% do tempo da CPU para escalonar seus processos com base no algoritmo FCFS [3].

3 Trabalhos Relacionados

3.1 Sistema de Correção do CD-MOJ

O sistema de correção do CD-MOJ gira em torno principalmente de 4 *scripts*:

1. `corrige.sh`[1]
2. `job-receiveitor.sh`
3. `root-daemon.sh`
4. `buid-and-test.sh`[2]

O loop principal do `corrige.sh` realiza o gerenciamento contínuo das submissões de problemas do CD-MOJ, utilizando os diretórios SUBMISSIONDIR (local onde estão armazenados os arquivos de submissões) e SUBMISSIONDIR-enviaroj (uma pasta específica para os arquivos prontos para submissão), o *script* processa periodicamente os arquivos presentes em SUBMISSIONDIR-enviaroj. Extrai dados relevantes dos nomes dos arquivos, como informações sobre os concursos, IDs dos problemas e linguagens de programação e submete as soluções para correção e monitora em segundo plano o *status* dessas submissões. Esse script chama outros que dependendo da situação ou envia direto para o SPOJ (`enviar-spoj.sh`) ou para o CD-MOJ (`enviar-cdmoj.sh`).

O `job-receiveitor.sh`, nesse caso, que foi acionado pelo `enviar-cdmoj.sh` funciona como um servidor TCP/IP (*Transmission Control Protocol/Internet Protocol*) que recebe solicitações externas contendo os pedidos de correção de problemas. Ele enfileira essas solicitações para processamento, preparando arquivos JSON (*JavaScript Object Notation*) contendo detalhes sobre os problemas, como *status*, código a ser corrigido, informações do sistema e a linguagem de programação associada.

Além disso, o `job-receiveitor.sh` executa comandos para adicionar novos *jobs* de correção, lista problemas disponíveis, verifica bloqueios na máquina, relata tempos-limite

[1]Disponível em: https://github.com/cd-moj/cdmoj/blob/master/server/judge/corrige.sh
[2]Disponível em: https://github.com/cd-moj/mojtools/blob/master/build-and-test.sh

dos problemas e detalhes da máquina. Também atualiza o conjunto de problemas, obtém informações precisas sobre problemas específicos e recupera resultados detalhados de *jobs* em execução. Em caso de comandos inválidos, fornece uma resposta de erro adequada.

Já no `buid-and-test.sh` é onde de fato ocorre a correção. Nele ocorre a definição de variáveis de diretórios e caminhos para o compilador da linguagem utilizada na submissão, estabelecendo os limites de recursos como memória e tempo de execução. Além disso, ele compila, executa e avalia as soluções dos usuários em relação aos testes definidos, gerando um veredito final com base nos resultados obtidos durante a execução do código submetido.

O processo é iniciado com a compilação do código-fonte submetido, onde este é copiado para um diretório temporário de trabalho e compilado usando um *script* específico da linguagem associada ao problema proposto; na ausência desse *script*, é utilizado o *script* padrão da linguagem. Em seguida, é executado uma bateria de testes previamente definidos no diretório do problema, estabelecendo limites de tempo e memória para cada teste, criando processos de execução do código submetido e registrando o desempenho, erros ou quaisquer problemas encontrados.

Por fim, os resultados de cada teste são avaliados, determinando se a solução atende aos critérios estabelecidos, gerando um veredito final e produzindo um relatório detalhado que indica se a solução foi aceita, caso houver erros, tempo limite excedido, entre outros aspectos avaliados durante a execução do código submetido.

3.2 Sistema de Correção do DMOJ

O DMOJ [4] é uma plataforma de competições de programação e um repositório de problemas, totalmente código aberto. Ele abriga problemas de competições anteriores e de diversas fontes. O DMOJ proporciona uma experiência moderna e acessível para entusiastas de programação. Os dados, incluindo problemas, submissões, usuários e competições, são disponibilizados em tempo real, enriquecendo a comunidade com uma ampla fonte de informações.

Para base de comparação e estudos, foi estudado de maneira geral o funcionamento do módulo de *Judge*[3] do DMOJ, responsável por corrigir as submissões e fornecer os resultados. Quando uma submissão é recebida, o processo principal da *Judge* cria um novo *JudgeWorker* para lidar com essa submissão específica.

[3]Disponível em: https://github.com/DMOJ/judge-server/blob/master/dmoj/judge.py

No sistema de julgamento do DMOJ, as *threads* são utilizadas para gerenciar diferentes aspectos do processo de avaliação das submissões, permitindo a comunicação assíncrona entre o juiz e os processos de julgamento, coordenando a execução dos casos de teste, relatórios de status e tratamento de erros.

A classe `Judge` possui uma *thread* principal que é responsável por iniciar o processo de avaliação de uma submissão específica. Quando é iniciada uma avaliação, essa *thread* cria uma nova *thread* para lidar com a execução da avaliação propriamente dita. Além disso, essa classe é responsável por coordenar todo o processo de avaliação. Ela gerencia a inicialização dos processos de julgamento, envia mensagens de controle para esses processos e lida com a comunicação entre o juiz e os *workers* durante a avaliação das submissões.

Essa nova *thread* de avaliação fica aguardando mensagens de comunicação vindas dos processos de julgamento (*JudgeWorker*). Essas mensagens são recebidas assincronamente por meio de um mecanismo de IPC (*Inter-Process Communication*), permitindo a troca de informações sobre o status da avaliação, resultados dos casos de teste, erros de compilação, entre outros.

Já a classe `JudgeWorker`, que também possui uma *thread* própria, representa os processos de julgamento. Cada instância dessa classe é responsável por executar a avaliação de uma submissão específica. Eles se comunicam com o juiz por meio de mensagens IPC, trocando informações sobre o estado da avaliação, resultados dos casos de teste, erros de compilação, entre outros

3.3 Sistema de Correção do BOCA

O BOCA [5] (BOCA Online Contest Administration) é um software desenvolvido em PHP para gerenciar competições de programação, semelhantes à Maratona de Programação da SBC. Ele permite que equipes interajam por meio de um navegador web, oferecendo diversas funcionalidades durante a competição.

Quando um participante submete uma solução, o sistema coleta os dados do envio, como o código-fonte, o problema associado e o idioma escolhido. Esses dados são processados, armazenados no banco de dados e posteriormente recuperadas para serem processadas pelo sistema. É realizada a exportação de arquivos, como códigos-fonte e pacotes de problemas, para utilização no processo de julgamento das submissões. Após a avaliação das submissões, os resultados são registrados no banco de dados, indicando se a submissão foi aprovada ou reprovada nos testes.

O funcionamento principal do sistema BOCA é observado em um *loop* que verifica a presença de tarefas a serem processadas no contexto de concursos ativos. Quando uma tarefa é identificada, o sistema executa uma série de etapas. O sistema BOCA executa fases distintas para compilação e execução dos códigos submetidos pelos participantes. Isso inclui a verificação de erros de compilação e a execução dos programas submetidos com os casos de teste fornecidos.

Durante a execução, o sistema compara as saídas dos programas submetidos com as saídas esperadas, determinando possíveis erros ou inconsistências. O sistema também monitora o uso de recursos como memória e tempo de execução para garantir a conformidade com os limites estabelecidos.

Ao final de cada fase de avaliação, o sistema BOCA registra os resultados, podendo sinalizar erros de compilação, problemas na execução ou discrepâncias nos resultados obtidos em relação aos esperados. Esses registros são essenciais para a classificação e pontuação dos participantes com base na precisão e correção de suas submissões.

4 Escalonador de Múltiplas Filas do CD-MOJ

Para a implementação de um sistema de escalonamento por múltiplas filas, o script `enviar-cdmoj.sh` foi atualizado e o `job-receiveitor.sh` foi modificado para dar espaço ao novo `job-receiveitor-master.sh` e também foi introduzido um novo script chamado `escalonador.sh`, que contém a lógica central da solução proposta. Adicionalmente, no `contest-description.txt`, foi inserido uma variável para designar máquinas específicas para as correções das submissões e outra variável que define a prioridade inicial de cada *contest* e direciona as submissões para a fila apropriada. Nas seções a seguir, cada modificação e implementação é abordada com detalhes.

4.1 Scripts

4.1.1 Contest Description

O arquivo `contest-description.txt` é essencial e deve seguir um modelo específico, uma vez que fornece uma descrição completa do *contest*, incluindo detalhes dos problemas, regras, datas importantes e critérios de avaliação. As novas variáveis são:

1. **`MOJCONTESTSERVERS:`** Esta variável permite direcionar as correções das submissões exclusivamente para uma ou mais máquinas específicas. Caso essa variável não seja fornecida, será considerado que o *contest* pode ser corrigido por qualquer máquina.
2. **`CONTEST_TYPE:`** Define o tipo de *contest*, direcionando-o para a fila de prioridade adequada. Caso essa variável não seja fornecida ou esteja vazia, será considerada a prioridade intermediária de lista-privada. Os tipos possíveis são, em ordem da maior para a menor prioridade:
   (a) super
   (b) prova
   (c) lista-privada
   (d) lista-publica

O exemplo de um arquivo preenchido com as novas variáveis:

```
...
MOJCONTESTSERVERS="gpu2 gpu3"
CONTEST_TYPE=prova
```

Essas configurações ajudam a garantir que os recursos sejam alocados corretamente e que as submissões sejam processadas de acordo com as necessidades específicas do *contest*.

4.1.2 Enviar CD-MOJ

No script `enviar-cdmoj.sh`, ocorreram mudanças significativas para aprimorar a comunicação e gestão das submissões. Inicialmente, foi alterada a porta HTTP para comunicação com o `job-receiveitor-master.sh`, centralizando a atribuição dos *jobs* às filas de prioridade. Além disso, adicionou-se lógica ao script para ler e processar as novas variáveis `MOJCONTESTSERVERS` e `CONTEST_TYPE`, que definem, respectivamente, os servidores específicos para processamento das submissões e o tipo de *contest*, influenciando diretamente na priorização das tarefas. Estas modificações visam melhorar a eficiência e a precisão no gerenciamento das submissões.

4.1.3 Job Receiveitor Master

O *script* `job-receiveitor-master.sh` é uma evolução do `job-receiveitor.sh`, desenvolvido para ser a principal interface de comunicação entre a interface do CD-MOJ e as máquinas de correção. Este *script* apresenta quatro funcionalidades principais:

1. `run`: Esta função determina o diretório apropriado para a submissão com base na variável `CONTEST_TYPE`, que indica a fila de prioridade. Ela cria um arquivo JSON com as informações da submissão, nomeado seguindo o padrão `TIMESTAMP_JOBID`. Isso garante um identificador único para cada submissão e estabelece uma ordem cronológica de chegada dentro da fila de prioridade.

2. `getresultindirect`: Esta função é responsável por recuperar resultados indiretos das submissões. Ela localiza o arquivo JSON no diretório de submissões enviadas, e extrai informações como *host*, porta e ID de execução, usando o JOBID como referência. Estes dados são utilizados nas funções `getresult` e `getresultfull` para facilitar a recuperação dos resultados.

3. `getresult`: Utiliza o *host*, a porta e o ID de execução para enviar uma requisição ao servidor onde a submissão foi corrigida. Esta função é destinada a obter o resultado

final direto de uma submissão, incluindo o *status* da correção e a data em que o *job* foi processado.

4. `getresultfull`: Funciona de maneira similar ao `getresult`, mas oferece um conjunto mais detalhado de informações. Além dos resultados da correção, esta função fornece logs de execução completos, que são especialmente úteis para serem exibidos no *bot* do Mojinho no Telegram.

Essas funções asseguram uma gestão eficaz e dinâmica das submissões, melhorando a comunicação e a eficiência do processamento dentro do sistema CD-MOJ.

4.1.4 Escalonador

O script `escalonador.sh` é a peça central do sistema de escalonamento. Este código opera continuamente dentro de um *loop* infinito, onde são executadas várias funções críticas para o gerenciamento eficiente das submissões. As funções são as seguintes:

1. `win_queue`: Esta função recursiva é projetada para gerenciar a fila de submissões que estão prontas para serem enviadas, mas que, por algum motivo externo, como uma falha ou queda de energia, não foram processadas. Primeiramente, a função utiliza `get_free_machine` para identificar quais máquinas estão disponíveis. Em seguida, executa `enviar_cdmoj` para tentar enviar os *jobs* pendente na fila de acordo com o número de máquinas livres. Após isso, ela revisita a fila para verificar se ainda existem *jobs* pendentes. Se houver, a `win_queue` é chamada novamente; se não, o fluxo retorna ao *loop* principal do escalonador.

2. `get_free_machine`: Esta função checa a disponibilidade das máquinas de correção. Enquanto a lista de máquinas livres estiver vazia, ela percorre todos os endereços das máquinas, enviando uma requisição para cada uma para verificar seu estado atual. Se uma máquina estiver disponível, ela é adicionada à variável `lista de livres`, que mantém o registro das máquinas prontas para receber novos *jobs*.

3. `run`: Esta função tem como objetivo otimizar a utilização das máquinas disponíveis, atribuindo *jobs* às máquinas livres de forma que todas estejam operacionais, caso possível. A função é estruturada em dois *loops* principais:

   (a) Primeiro *Loop*: Percorre as filas de prioridade para identificar *jobs* que requerem execução em uma porta específica, conforme indicado pela variável `MOJCONTESTSERVERS`. Se uma máquina específica estiver disponível para esse

job, ela é selecionada da lista de máquinas livres e imediatamente removida dessa lista para evitar sobreposição na atribuição de máquinas.

(b) Segundo *Loop*: Após tratar os *jobs* que requerem portas específicas, este *loop* revisita as filas para atribuir máquinas aos *jobs* que podem ser processados em qualquer máquina disponível. Cada *job* selecionado é atribuído à primeira máquina da lista de máquinas livres, que é então removida para garantir uma correta contabilização.

Todos os *jobs* selecionados são enviados para o diretório de enfileirados. Se o número total de *jobs* nas filas for menor que o número de máquinas disponíveis, todos os *jobs* disponíveis serão processados sem deixar nenhuma máquina ociosa. Este método visa garantir uma maior capacidades de processamento, maximizando o uso das máquinas e minimizando o tempo de inatividade.

4. `enviar_cdmoj`: Esta função é responsável por gerenciar o envio de *jobs* para as máquinas designadas. Ela seleciona *jobs* do diretório de enfileirados, baseando-se no número de máquinas disponíveis. Para cada *job*, a função extrai informações cruciais do arquivo JSON associado, incluindo o *host* e a porta específicos para onde o *job* deve ser enviado, além de detalhes como o ID do problema, a linguagem de programação utilizada e o arquivo de submissão codificado em base64. Em seguida, esses dados são transmitidos para a máquina especificada para que a correção seja realizada. Após o envio, o *job* é movido para o diretório de enviados, garantindo que o sistema mantenha um registro organizado das submissões processadas.

5. `check_starvation`: Este método desempenha uma função crucial ao monitorar e prevenir o fenômeno de *starvation* nas filas de prioridades. Ele percorre todos os *jobs* nas diversas filas e analisa o tempo que cada *job* tem estado na fila, comparando o *timestamp* atual com o *timestamp* indicado no nome do arquivo do *job*. Se um *job* estiver na fila por mais de 300 segundos, ele é automaticamente promovido para uma fila de prioridade superior, caso contrário, o *job* permanece na sua fila atual. Este procedimento ajuda a manter um fluxo de trabalho eficiente e igualitário, assegurando que todos os *jobs* sejam processados sem atrasos desproporcionais.

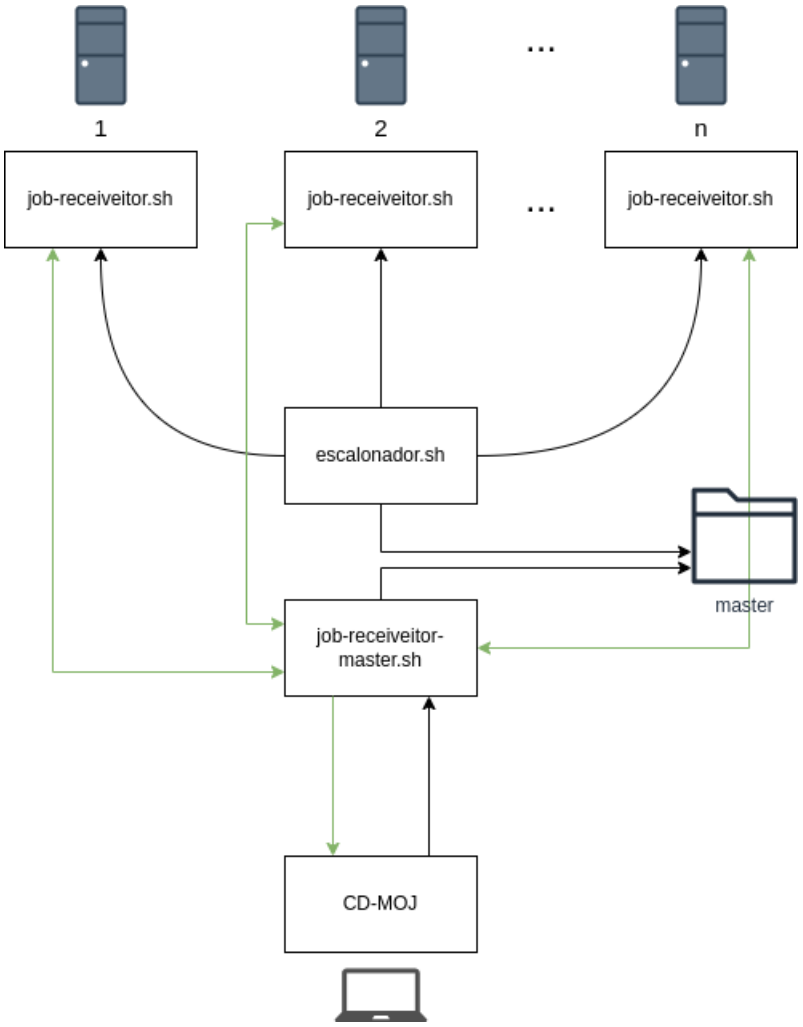
Este ciclo contínuo de operações no `escalonador.sh` facilita um fluxo constante e eficiente de processamento das submissões, contribuindo para a otimização geral do sistema.

4.2 Funcionamento

4.2.1 Visão Geral

Quando um usuário envia uma submissão no sistema CD-MOJ, o processo segue o fluxo descrito abaixo:

Figura 4.1. Visão geral do sistema de escalonamento de submissões do CD-MOJ



Fonte: Autor

Inicialmente, o *script* de enviar submissão do CD-MOJ faz uma requisição para o servidor TCP do `job-receiver-master.sh`, que é responsável por alocar o *job* na fila apropriada dentro do diretório `master`. Em paralelo, o `escalonador.sh` opera em um *loop* infinito, executando a cada segundo as funções necessárias para gerenciar as filas de submissões.

Após ser alocado na fila correta, o *job* é movido para a fila de `enfileirados` do diretório `master` e enviado para o `job-receiver.sh` de uma máquina disponível, onde o

job é processado. Após a correção, o *job* é transferido para a fila de `enviados` do diretório `master`. Como o resultado da submissão não é retornado imediatamente, o *script* de envio executa um *pulling* no `job-receiver-master.sh`, que por sua vez envia uma requisição para obter o resultado específico do *job* processado.

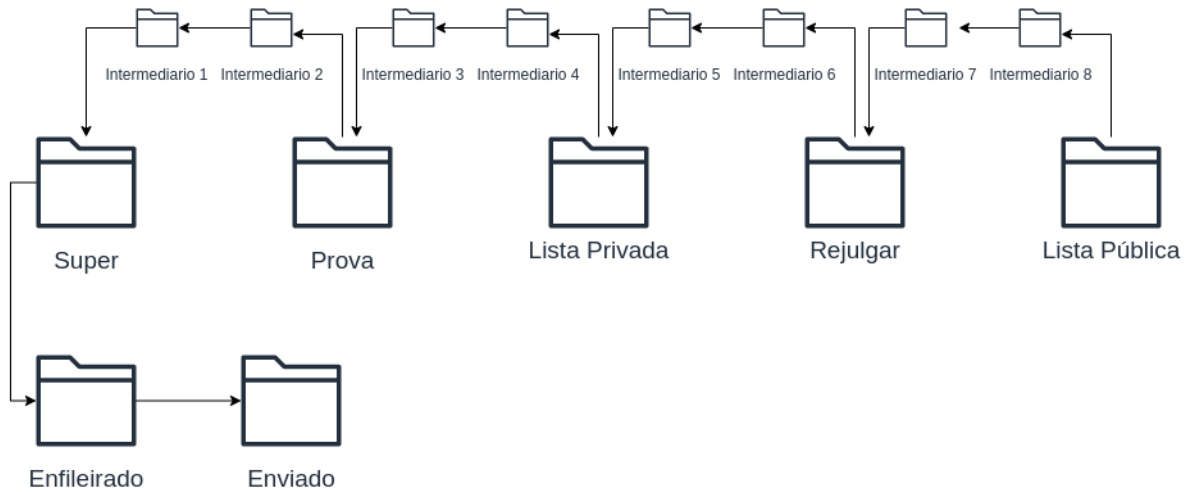
Finalmente, o resultado é retornado ao sistema CD-MOJ e disponibilizado ao usuário. Este processo garante uma gestão eficaz e ordenada das submissões, minimizando o tempo de espera e maximizando a eficiência do sistema.

4.2.2 Fila de Prioridades

O diretório `master` desempenha um papel essencial, não apenas armazenando diretórios de *log* e arquivos de *lock*, mas também organizando 13 diretórios que correspondem às filas de prioridades. Destes, cinco são diretórios principais — `super`, `prova`, `lista-privada`, `rejulgar`, `lista-publica` — dos quais quatro podem ser selecionados pelo autor do *contest* conforme a necessidade. Os 8 diretórios intermediários são usados para mitigar a rápida ascensão de prioridades, uma medida preventiva crucial durante períodos de intenso volume de submissões.

Além disso, o diretório `rejulgar` é selecionado automaticamente quando o autor do *contest* opta por rejulgar todas as submissões enviadas ao sistema. Há também dois outros diretórios significativos: o `enfileirados` que é destinado a abrigar os *jobs* selecionados antes de serem enviados para correção e o `enviados` para as submissões que já foram processadas. A estrutura e o fluxo operacional dentro do diretório master podem ser visualizados na Figura 4.2.

Figura 4.2. Diretório Master com as Filas de Prioridades



Fonte: Autor

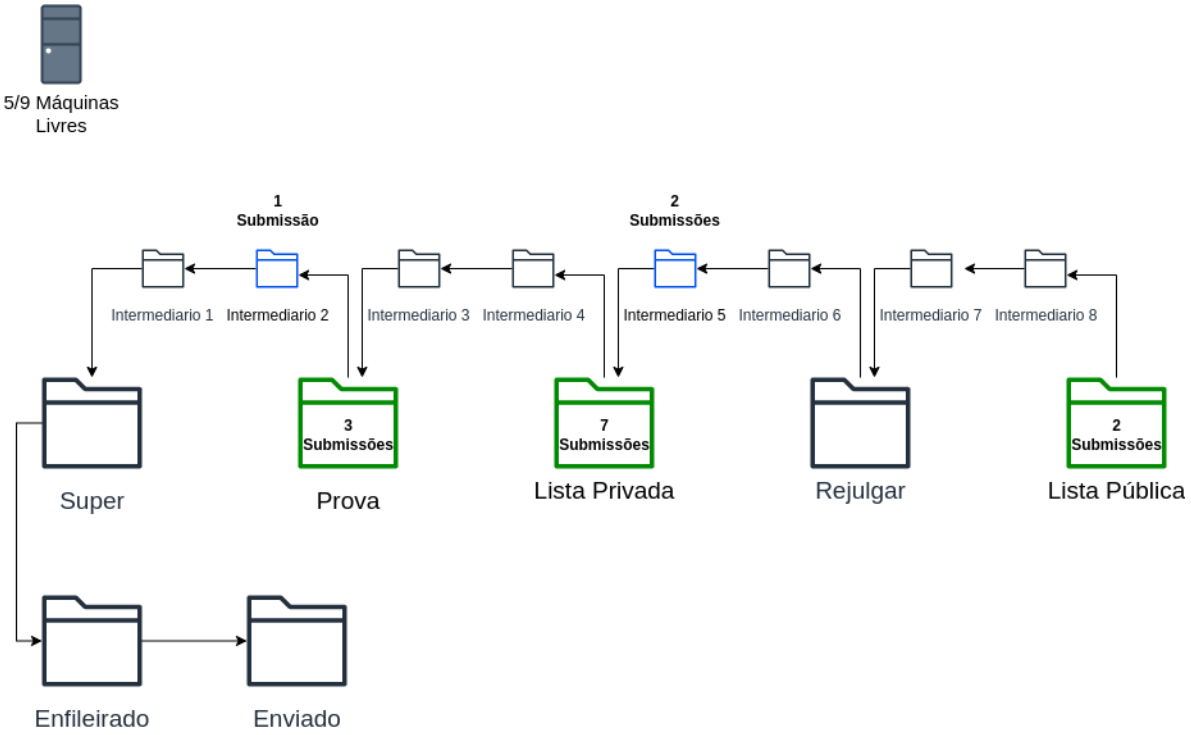
Para exemplificar o funcionamento do ciclo do escalonador, considere a situação descrita na Figura 4.3. Suponha que existam cinco máquinas disponíveis de um total de nove e a distribuição das submissões pelas filas seja a seguinte: uma submissão no diretório `intermediário 2`, três em `prova`, `sete` em `lista privada`, duas no `intermediário 5` e duas em `lista pública`. Neste cenário, o escalonador está configurado para selecionar apenas cinco submissões, respeitando a capacidade das máquinas disponíveis.

Durante o primeiro ciclo do escalonador, ele percorre os diretórios de prioridades da esquerda para a direita, selecionando as submissões na seguinte ordem:

- Uma submissão do diretório `intermediário 2`.
- Três submissões do diretório de `prova`.
- Uma submissão do diretório de `lista privada`, completando as cinco máquinas disponíveis.

Ao fim deste ciclo, as submissões restantes nos diretórios de `lista privada`, `intermediário 5` e `lista pública` aguardarão o próximo ciclo do escalonador para serem potencialmente selecionadas.

Figura 4.3. Exemplo 1: Funcionamento das Filas de Prioridades



Fonte: Autor

No segundo ciclo, que ocorre imediatamente após o primeiro, as condições são ilus-

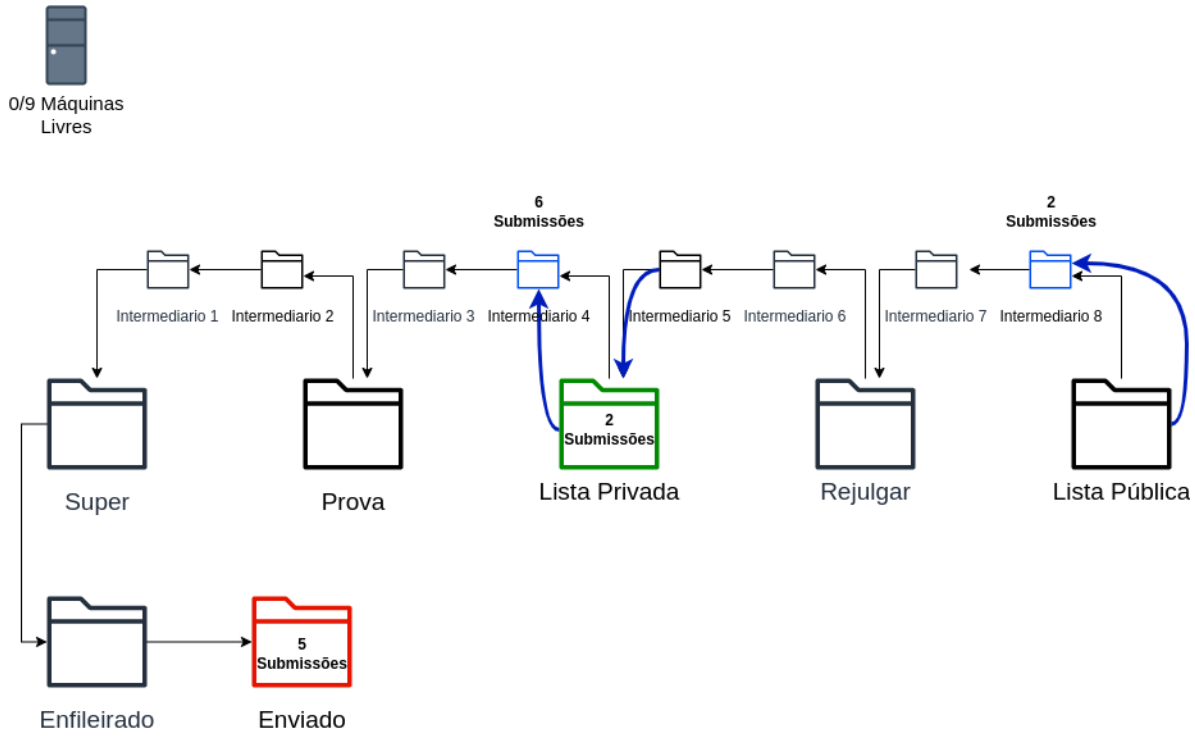
tradas na Figura 4.4. Neste momento, todas as máquinas estão ocupadas corrigindo exercícios que requerem um tempo de correção prolongado. Simultaneamente, todas as submissões ainda disponíveis ultrapassaram o limite de 300 segundos de espera nos seus respectivos diretórios, sendo então promovidas automaticamente para o diretório de nível superior.

A promoção é realizada também com uma atualização do *timestamp* no nome de cada arquivo, essencial para reiniciar o contador de tempo de espera em cada novo diretório. Essa atualização assegura que as submissões não sejam promovidas novamente no ciclo subsequente sem terem completado os 300 segundos necessários no novo diretório.

Analisando a movimentação das submissões da direita para a esquerda na figura:

- As submissões do diretório de lista pública são movidas para o `intermediário 8`.
- As duas submissões do `intermediário 5` são promovidas para o diretório de `lista privada`.
- As seis submissões restantes no diretório de `lista privada` são transferidas para o `intermediário 4`.

Figura 4.4. Exemplo 2: Funcionamento das Filas de Prioridades



Fonte: Autor

5 Resultados

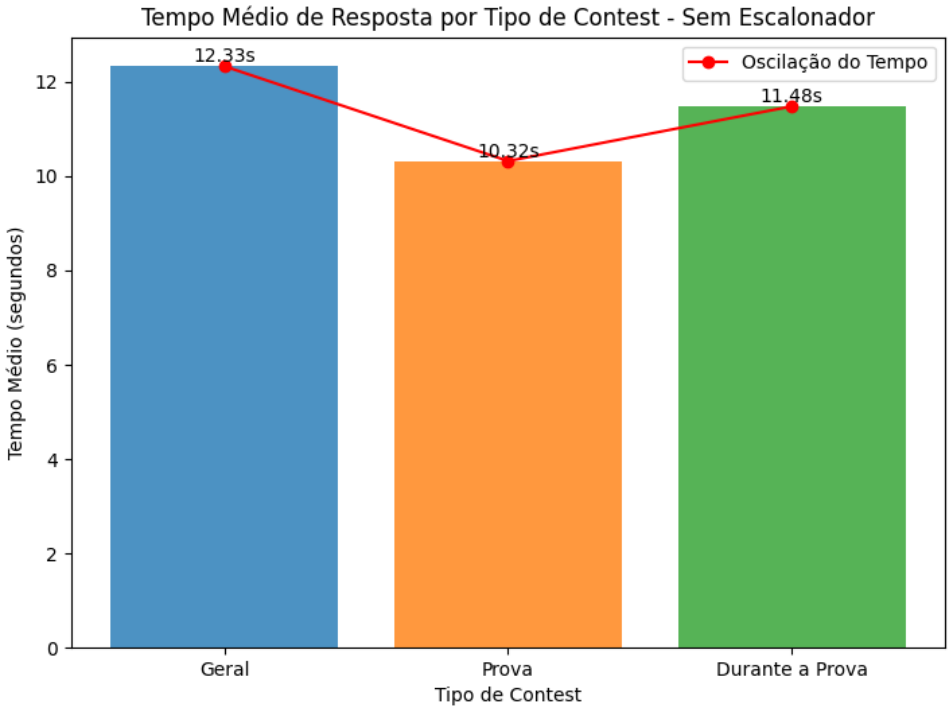
Este capítulo apresenta a análise dos tempos de resposta médios antes e depois da implementação do escalonador, com foco em entender as melhorias trazidas por esta intervenção. Serão detalhados os resultados obtidos durante o período de março a junho de 2024, abrangendo tanto submissões gerais quanto específicas, como provas e listas mais complexas.

5.1 Antes da Implementação do Escalonador

Durante o período de março a junho de 2024, foram analisadas um total de 44.250 submissões. O tempo médio de resposta foi de 12,33 segundos, representando o intervalo desde a submissão do exercício pelo aluno até o recebimento do *feedback* da correção.

Em 26 de abril de 2024, durante uma prova de APC na FGA, foram processadas 454 submissões, com um tempo médio de resposta de 10,32 segundos. No período de total dessa prova, entre 10h10 e 12h30, foram processadas 582 submissões, 128 sendo de outras listas, com um tempo médio de resposta total de 11,48 segundos. A Figura 5.1 a seguir ilustra a análise do tempo total médio de resposta sem o escalonador.

Figura 5.1. Análise do Tempo Total Médio sem o Escalonador



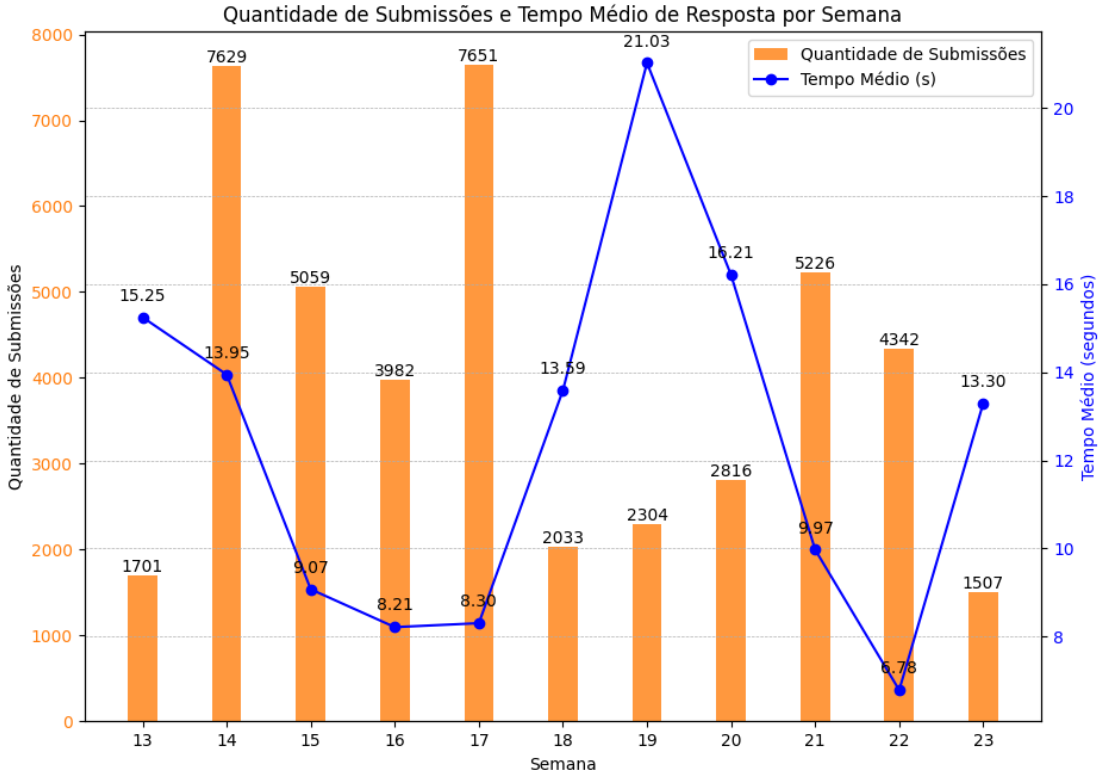
Fonte: Autor

Observa-se um tempo de resposta significativamente menor para as provas em comparação com os contests em geral. Essa diferença pode ser atribuída ao horário específico em que as provas são realizadas, à demanda de submissões nesses períodos e a complexidade dos exercícios.

5.1.1 Semanal

Para fins comparativos, a Figura 5.2 apresentada a seguir ilustra o tempo médio total de correção para todas as semanas do primeiro semestre letivo de 2024, antes da implementação do escalonador. Esta análise permite observar o comportamento do tempo de resposta ao longo das semanas, destacando como variações na quantidade e na complexidade dos exercícios afetam esse tempo.

Figura 5.2. Análise do Tempo Total Médio e Submissões por Semana



Fonte: Autor

A semana de maior atividade foi a semana 17, na qual foram registradas 7.651 submissões, com um tempo médio total de correção de 8,30 segundos. Em contraste, a semana com menor atividade, a semana 23, compreendendo apenas os três primeiros dias e sendo a última semana antes da implementação do escalonador, apresentou um tempo médio de correção de 13,30 segundos.

5.2 Depois da Implementação do Escalonador

A implementação do escalonador, iniciada em 6 de junho de 2024, envolveu a análise de 10.304 submissões até o dia 30 de junho de 2024. Esse processo resultou em uma melhoria significativa na gestão dos tempos de resposta. A análise detalhada do tempo na fila e do tempo de correção permitiu uma redução significativa no tempo médio total de resposta.

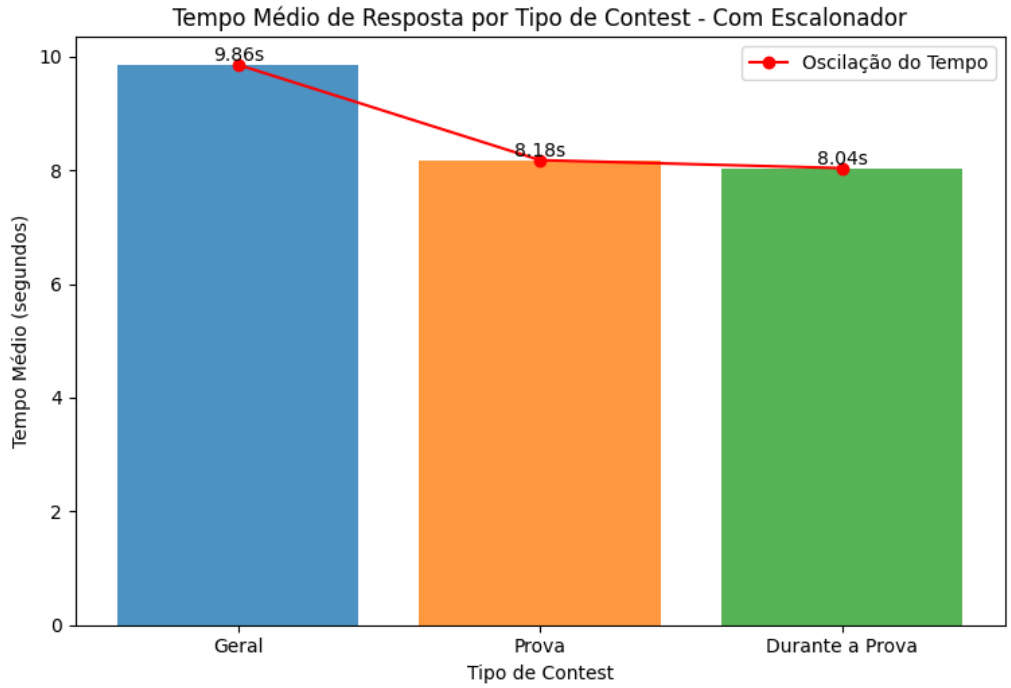
Durante o período após a implementação do escalonador, observou-se uma diminuição de 2,47 segundos no tempo médio de resposta. Além disso, foi possível analisar separada-

mente quanto tempo cada submissão gastou na fila até entrar para a correção e quanto tempo foi necessário para a correção, detalhado a seguir:

- **Tempo na fila:** 4,51 segundos.
- **Tempo em correção:** 5,35 segundos.
- **Tempo médio de resposta total:** 9,86 segundos (representando uma redução de 20,03% em relação à média geral antes do escalonador).

Em 21 de junho de 2024, durante a segunda prova de APC na FGA, foram processadas 477 submissões, com um tempo médio de resposta de 8,18 segundos. No período total dessa prova, entre 10h10 e 12h30, foram processadas 489 submissões, sendo 12 de outras listas, com um tempo médio de resposta total de 8,04 segundos. A Figura 5.3 a seguir ilustra a análise do tempo total médio de resposta com o escalonador.

Figura 5.3. Análise do Tempo Total Médio com o Escalonador



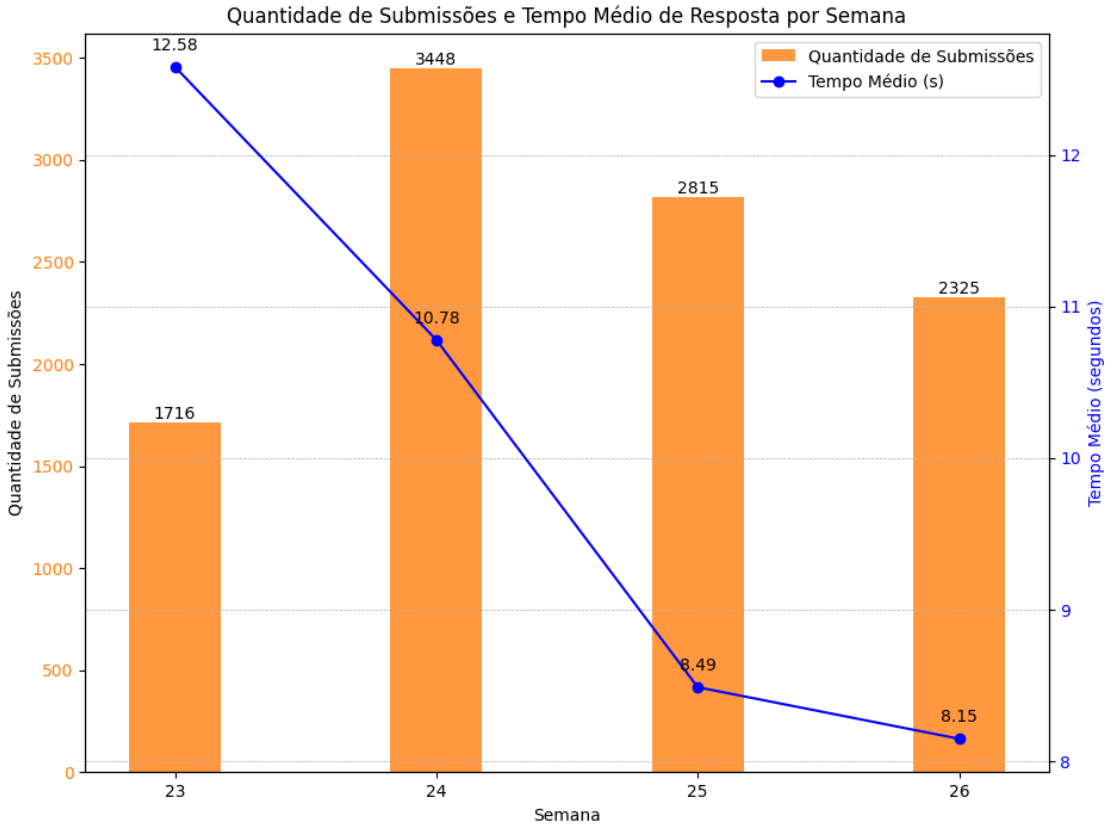
Fonte: Autor

5.2.1 Semanal

Para fins comparativos, a Figura 5.4 apresentada a seguir ilustra o tempo médio total de correção semanal após a implementação do escalonador. Esta análise permite observar

o comportamento do tempo de resposta ao longo das semanas, destacando como variações na quantidade e na complexidade dos exercícios afetam esse tempo.

Figura 5.4. Análise do Tempo Total Médio e Submissões por Semana - Com o Escalonador



Fonte: Autor

A semana de maior atividade foi a semana 24, na qual foram registradas 3.448 submissões, com um tempo médio total de correção de 10,78 segundos. Em contraste, a semana com menor atividade, correspondente à semana 23 e abrangendo apenas os dois últimos dias, foi a primeira semana após a implementação do escalonador, apresentando um tempo médio de correção de 12,58 segundos.

5.3 Comparação

A Tabela 5.1 apresenta a análise dos tempos médios individuais por estágio, comparando o desempenho do sistema com e sem o escalonador. O tempo de correção foi mantido constante em ambos os cenários, uma vez que nada foi alterado na etapa de

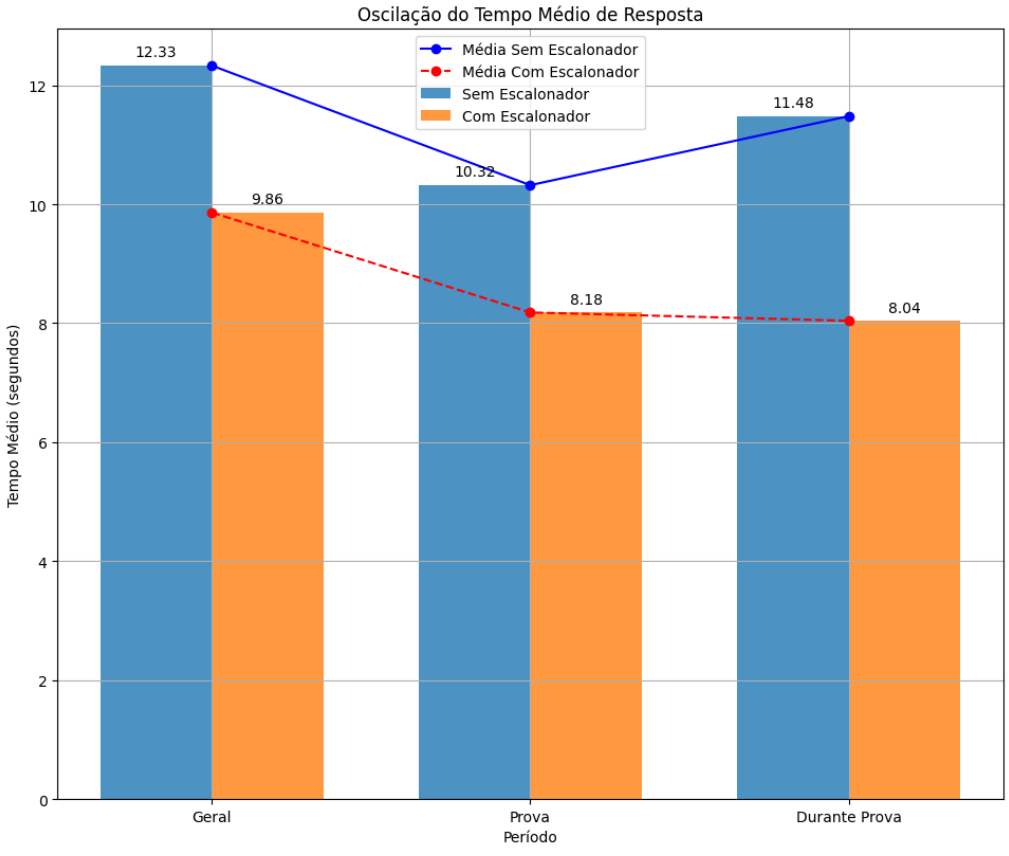
correção das submissões, apenas no escalonador. Ou seja, como após a implementação do escalonador é possível saber quanto tempo cada *job* permanece na fila, é possível também determinar quanto tempo ele passa em correção. Com isso em vista, é possível uma comparação direta entre os tempos na fila e os tempos médios de resposta, conforme detalhado na Tabela 5.1.

Tabela 5.1. Tempo em Diferentes Estágios com e sem Escalonamento

Estágio	**Sem Escalonador (s)**	**Com Escalonador (s)**	**Redução (%)**
Tempo na fila	6,98	4,51	35,39
Tempo em correção	5,35	5,35	0
Tempo Médio de Resposta	12,33	9,86	20,03

Os resultados demonstram a eficácia do escalonador na otimização do tempo de cada submissão na fila, resultando em uma redução significativa do tempo médio de resposta e melhorando a experiência dos usuários. A Figura 5.5 reúne os principais tempos médios sem e com o escalonador.

Figura 5.5. Oscilação do Tempo Médio de Resposta



Fonte: Autor

A Figura 5.5 ilustra a redução significativa nos tempos de resposta do sistema de correção de exercícios após a implementação do escalonador. A análise detalhada dos tempos na fila e de correção permitiu a identificação de pontos de melhorias no sistema e a otimização do processo, resultando em um sistema mais eficiente e responsivo.

6 Conclusão

Este trabalho descreveu detalhadamente a implementação de um sistema de escalonamento baseado em múltiplas filas de prioridade o CD-MOJ. O principal objetivo deste trabalho foi aperfeiçoar a eficiência do sistema de correção de exercícios e a alocação de recursos computacionais através da implementação de um escalonador de submissões. Conforme demonstrado no Capítulo 5, os resultados obtidos foram notavelmente positivos.

A implementação resultou em uma redução significativa no tempo médio de resposta para as submissões em geral, corroborando a eficácia do sistema de múltiplas filas de prioridade. Além disso, a definição de prioridades específicas para submissões como as do tipo `prova` provou ser crucial, uma vez que, como observado, beneficiou-se de uma priorização notável. Isso não apenas melhorou a experiência dos usuários, mas também otimizou o uso dos recursos computacionais disponíveis.

Em suma, este estudo confirma a importância de escalonar submissões, uma vez que cada tipo de *contest* possui um propósito distinto. Com a adição de novas funcionalidades, torna-se possível delegar máquinas específicas para cada *contest*, otimizando o uso dos recursos. Além disso, o sistema de escalonamento permite minimizar períodos de ociosidade das máquinas, assegurando que máquinas livres sejam eficientemente aproveitadas em cada ciclo de execução.

Diversas melhorias podem ser incorporadas ao sistema de escalonamento atual para aumentar ainda mais sua eficácia e eficiência. Algumas sugestões incluem:

- **Preempção de Submissões e Retomada Automática:** A preempção de submissões envolve interromper processos de menor prioridade para dar lugar a processos de maior urgência. Implementar essa funcionalidade poderia garantir que recursos críticos sejam alocados de forma mais eficiente, especialmente em períodos de alta demanda. Além disso, a retomada automática de execuções interrompidas asseguraria a continuidade e a integridade do processo de correção, sem perder o progresso realizado.

- **Paralelização da Correção de Casos de Teste:** Aproveitando os diversos núcleos disponíveis na CPU, a paralelização da correção de casos de teste poderia reduzir significativamente o tempo total de correção. Atualmente, cada caso de teste é processado de forma linear em uma única máquina, mas ao distribuir essas tarefas simultaneamente entre vários núcleos da CPU, o sistema poderia diminuir o tempo de espera e aumentar a capacidade de processamento.

Essas melhorias não apenas otimizariam o uso de recursos computacionais, como também melhorariam a experiência do usuário, reduzindo o tempo de espera e aumentando a eficiência do sistema de correções.

Referências Bibliográficas

[1] CD-MOJ. Contest driven meta online judge. 2013.

[2] Andrew S. Tanenbaum e Herbert Bos. *Sistemas Operacionais Modernos*. Pearson Education do Brasil, São Paulo, 4ª edição, 2016. Tradução Jorge Ritter; revisão técnica Raphael Y. de Camargo.

[3] Abraham Silberschatz e Greg Galvin, Peter Baer e Gagne. *Fundamentos de Sistemas Operacionais*. LTC, Rio de Janeiro, 9ª edição, 2015.

[4] DMOJ - Don Mills Online Judge. Plataforma de Julgamento Online. https://dmoj.ca/, s/d. Acesso em: 20 de novembro de 2023.

[5] Cassio P de Campos e Carlos E Ferreira. Boca: um sistema de apoio a competições de programação. In *Workshop de Educação em Computação*. Sociedade Brasileira de Computação, 2004.