Relatório Técnico: Otimização de Escalonamento de Tarefas em Sistemas Computacionais

Sistema de Gerenciamento de Tarefas

24 de Novembro de 2025

Resumo

Este documento apresenta uma análise detalhada do problema de gerenciamento de tarefas em sistemas computacionais com recursos limitados. Comparamos duas abordagens algorítmicas: uma heurística gulosa (*Weighted Job Scheduling*) e uma solução exata baseada em Programação Dinâmica (*0/1 Knapsack Problem*). O projeto destaca-se pela implementação manual de todos os algoritmos, incluindo o método de ordenação *Quick Sort*, e pela análise crítica da otimização de desempenho.

Conteúdo

1 Definição do Problema

1.1 Contextualização

O gerenciamento de tarefas em sistemas computacionais modernos é um desafio crítico. Um centro de processamento de dados recebe continuamente diversas tarefas, cada uma caracterizada por:

- **Tempo de Execução (t_i)**: Recurso temporal necessário para completar a tarefa.
- **Prioridade (v_i)**: Valor ou importância associada à conclusão da tarefa.
- **Prazo (d_i)**: Momento limite ou fator de urgência.

Como os recursos (tempo de processamento disponível T) são limitados, torna-se impossível executar todas as tarefas simultaneamente. O problema consiste em selecionar um subconjunto de tarefas S tal que a soma dos tempos não exceda T e a soma das prioridades seja maximizada.

1.2 O Desafio

O desafio central reside em modelar uma estratégia de escalonamento que seja eficiente computacionalmente, mas que também evite decisões míopes que comprometam o desempenho global. A escolha entre uma resposta rápida (mas potencialmente subótima) e uma resposta ótima (mas computacionalmente mais custosa) é o cerne desta análise.

2 Implementação e Metodologia

2.1 Implementação Manual de Algoritmos

Em conformidade com os requisitos do projeto, **todos os algoritmos principais foram implementados manualmente**, sem o uso de bibliotecas de otimização ou funções de ordenação nativas (como `Collections.sort`). Isso demonstra o domínio sobre a lógica algorítmica e permite um controle refinado sobre o comportamento do sistema.

2.2 Método de Ordenação: Quick Sort

Para a etapa de pré-processamento (essencial para o algoritmo Guloso e organizacional para o DP), implementamos o algoritmo **Quick Sort**.

Justificativa da Escolha: Optamos pelo Quick Sort em detrimento de algoritmos mais simples como Bubble Sort ou Insertion Sort ($O(n^2)$) devido à sua eficiência média de $O(n \log n)$. Em um sistema de escalonamento real, onde o número de tarefas pode ser grande, a eficiência da ordenação é crucial para não se tornar um gargalo antes mesmo do processo de seleção começar. O critério de ordenação foi a **Densidade** ($\frac{\text{Prioridade}}{\text{Tempo} \times (\text{Prazo}+1)}$), ordenando as tarefas da mais "lucrativa" para a menos lucrativa.

2.3 Interface Humano-Computador (IHC)

O sistema foi desenvolvido com uma interface baseada em terminal (CLI - Command Line Interface). **Justificativa:** A escolha por uma interface textual permite foco total na lógica algorítmica e na visualização clara dos dados de saída (tabelas de comparação), sem o overhead de processamento gráfico. A saída é organizada e explicativa, facilitando a análise imediata dos resultados pelo operador.

3 Abordagem Proposta e Justificativa

Nosso sistema implementa e compara duas estratégias, mas defendemos a adoção do algoritmo baseado no **Problema da Mochila (Knapsack Problem)** em detrimento da abordagem puramente Gulosa.

3.1 Por que Knapsack (Programação Dinâmica)?

Embora o algoritmo Guloso seja intuitivo e extremamente rápido, ele sofre de uma falha fundamental: a incapacidade de "enxergar o futuro". A abordagem de Programação Dinâmica (DP) garante matematicamente a solução ótima global, avaliando implicitamente todas as combinações viáveis.

4 Documento Analítico

4.1 Formulação do Problema

Seja um conjunto de n tarefas, onde cada tarefa i tem um peso w_i (tempo) e um valor v_i (prioridade). Seja W a capacidade total. Queremos maximizar $\sum v_i$ sujeito a $\sum w_i \leq W$.

4.2 Modelo de Solução (Programação Dinâmica)

Definimos $dp[i][w]$ como o valor máximo considerando as primeiras i tarefas com capacidade w.

$$dp[i][w] = \max(dp[i-1][w], v_i + dp[i-1][w-w_i]) \quad \text{se } w_i \leq w$$

4.3 Análise Assintótica de Complexidade

4.3.1 1. Escalonador Guloso

- **Tempo**: $O(n \log n)$ (Ordenação Quick Sort) + $O(n)$ (Seleção) = $O(n \log n)$.
- **Espaço**: $O(n)$ (Lista e Pilha de Recursão).

4.3.2 2. Escalonador Knapsack (DP)

- **Tempo**: $O(n \cdot W)$. Preenchimento da tabela $n \times W$.
- **Espaço**: $O(n \cdot W)$. Armazenamento da tabela.

5 Análise de Desempenho Experimental

Realizamos testes empíricos comparando as duas abordagens.

5.1 Resultados Obtidos

Tabela 1: Comparativo de Valor Total e Tempo de Execução

Cenário	Algoritmo	Valor Total	Tempo Usado	Execução (ms)
1. Favorável	Guloso	37	10	≈ 0.15
(Limite: 10)	DP	37	10	≈ 0.20
2. Guloso Falha	Guloso	11	2	≈ 0.05
(Limite: 6)	**DP**	**20**	5	≈ 0.10
3. Idênticas	Guloso	60	30	≈ 0.08
(Limite: 30)	DP	60	30	≈ 0.15
4. Aleatório	Guloso	335	98	≈ 0.16
(Limite: 100)	**DP**	**352**	100	≈ 0.45

5.2 Discussão

O DP superou o Guloso em cenários críticos (Cenário 2 e 4), provando sua robustez. O tempo de execução do DP, embora maior, manteve-se na ordem de frações de milissegundo, o que é perfeitamente aceitável para a garantia de otimalidade. A ordenação Quick Sort manual comportou-se de forma estável e eficiente.

6 Análise Crítica Final

6.1 Adequação da Abordagem

Por que a abordagem escolhida foi a ideal? A escolha pela Programação Dinâmica foi ideal porque, no contexto de gerenciamento de tarefas críticas, a perda de valor (prioridade) é inaceitável. O algoritmo Guloso, apesar de rápido, falhou em maximizar o uso dos recursos em cenários de ”mochila cheia”, deixando lacunas de tempo que poderiam ser preenchidas por tarefas mais valiosas. O DP preencheu essas lacunas perfeitamente.

6.2 Metodologia e Trabalho em Equipe

Como o trabalho em equipe e a metodologia ágil influenciaram? O desenvolvimento seguiu uma abordagem incremental. Inicialmente, focamos na implementação da estrutura básica (‘Tarefa‘) e do algoritmo Guloso para ter um MVP (Minimum Viable Product). Em seguida, implementamos o DP para comparação. A divisão clara de responsabilidades (implementação de algoritmos vs. infraestrutura de testes) e o uso de testes automatizados (‘Comparador‘) permitiram identificar rapidamente as falhas do algoritmo Guloso, guiando a decisão final de recomendar o DP.

7 Conclusão

A implementação do algoritmo *Knapsack* via Programação Dinâmica, suportada por uma ordenação eficiente *Quick Sort*, provou ser a solução superior. O sistema atende a todos os requisitos de otimização, robustez e implementação manual, garantindo o melhor desempenho possível para o centro de processamento de dados.