Sistemas Operacionais 2/2025

Prof. Michel Junio Ferreira Rosa

Projeto Prático

BSB Compute: Orquestração de Tarefas

Davi Ribeiro Nobre - 22401443

Gabriella Lourenço - 22405748

Lara Alves Korogi - 22400272

Larissa Alves Neves - 22511353

1. Introdução

Com o crescimento de aplicações baseadas em Inteligência Artificial e processamento distribuído, tornou-se essencial modelar sistemas capazes de gerenciar requisições de forma eficiente, justa e organizada. Este projeto simula o funcionamento de um orquestrador distribuído, responsável por receber, priorizar e processar tarefas, seguindo políticas clássicas de escalonamento.

Ao contrário de implementações em baixo nível, este sistema foi criado inteiramente em Python, utilizando estruturas de dados, classes, módulos e manipulação de arquivos JSON para representar requisições, tarefas, métricas e resultados.

O objetivo do projeto é permitir ao estudante compreender:

- como funciona o fluxo de escalonamento;
- como organizar tarefas em múltiplas filas;
- como aplicar algoritmos diferentes (RR, SJF, Prioridades);
- como medir desempenho;
- como estruturar um simulador realista utilizando uma linguagem de alto nível

2. Objetivos do Projeto

2.1 Objetivo Geral

Desenvolver um sistema de simulação de orquestração capaz de processar requisições aplicando diferentes políticas de escalonamento, utilizando Python como linguagem principal.

2.2 Objetivos Específicos

- Implementar escalonamento Round Robin, SJF e Prioridades.
- Pré Processar dados de entrada via JSON.
- Simular processamento de tarefas com tempos variáveis.
- Gerar métricas no formato JSON.
- Salvar resultados em diretórios organizados.

- Desenvolver scripts de execução automática.
- Comparar cenários e tempos utilizando scripts auxiliares (comparador).

3. Arquitetura Geral do Sistema

A arquitetura é baseada em módulos Python, cada um com uma responsabilidade específica.
O sistema não utiliza processos do sistema operacional, mas sim simulação interna em Python para representar o comportamento de workers e do orquestrador.

Componentes principais:

3.1 main.py – Núcleo do Orquestrador

Contém:

- leitura do arquivo `config.json`
- carregamento das requisições
- aplicação do algoritmo escolhido
- simulação do processamento (com tempo via sleep)
- geração de métricas
- salvamento dos resultados no diretório `resultados/`

3.2 launcher.py – Controlador da Execução

Responsável por:

- iniciar as execuções
- selecionar qual algoritmo será utilizado
- chamar o módulo principal (main.py)
- automatizar múltiplas simulações
- garantir padronização de execução

3.3 comparador.py – Ferramenta de Análise

Executa:

- comparação de tempos médios
- comparação de diferentes algoritmos
- análises de desempenho
- geração de relatório final comparativo

3.4 config.json – Arquivo de Entrada

Contém:

- lista de requisições
- tipo da requisição
- tempo de execução
- prioridade
- parâmetros globais de simulação

3.5 resultados/ – Diretório de Saída

Contém arquivos JSON com:

- tempos médios de resposta
- ordem de execução
- logs da simulação
- desempenho por algoritmo

3.6 metricas.json

Arquivo de métricas consolidado criado após cada execução.

3.7 pyproject.toml

Gerencia as dependências do projeto e define como ele deve ser executado usando UV

3.8 uv.lock

Arquivo gerado pelo UV, registrando as versões exatas das dependências.

4. Tecnologias Utilizadas

4.1 Python 3

Linguagem principal do projeto.
Utilizada por vantagens como:

- fácil leitura
- suporte a orientação a objetos
- bibliotecas padrão suficientes para simulações
- excelente para manipulação de arquivos JSON
- rápida prototipação

4.2 Módulos internos do Python utilizados

- `json` — leitura e escrita de arquivos de configuração e resultados.
- `time` — simulação de tempo de execução (sleep).
- `random` — geração de tempos e cenários (se utilizado).
- `os` — manipulação de diretórios.
- `sys` — leitura de argumentos para escolher algoritmos

Utilizado tanto como entrada quanto como saída do sistema:

- entrada - config.json
- saída - metricas.json e resultados/

4.5 Scripts Python

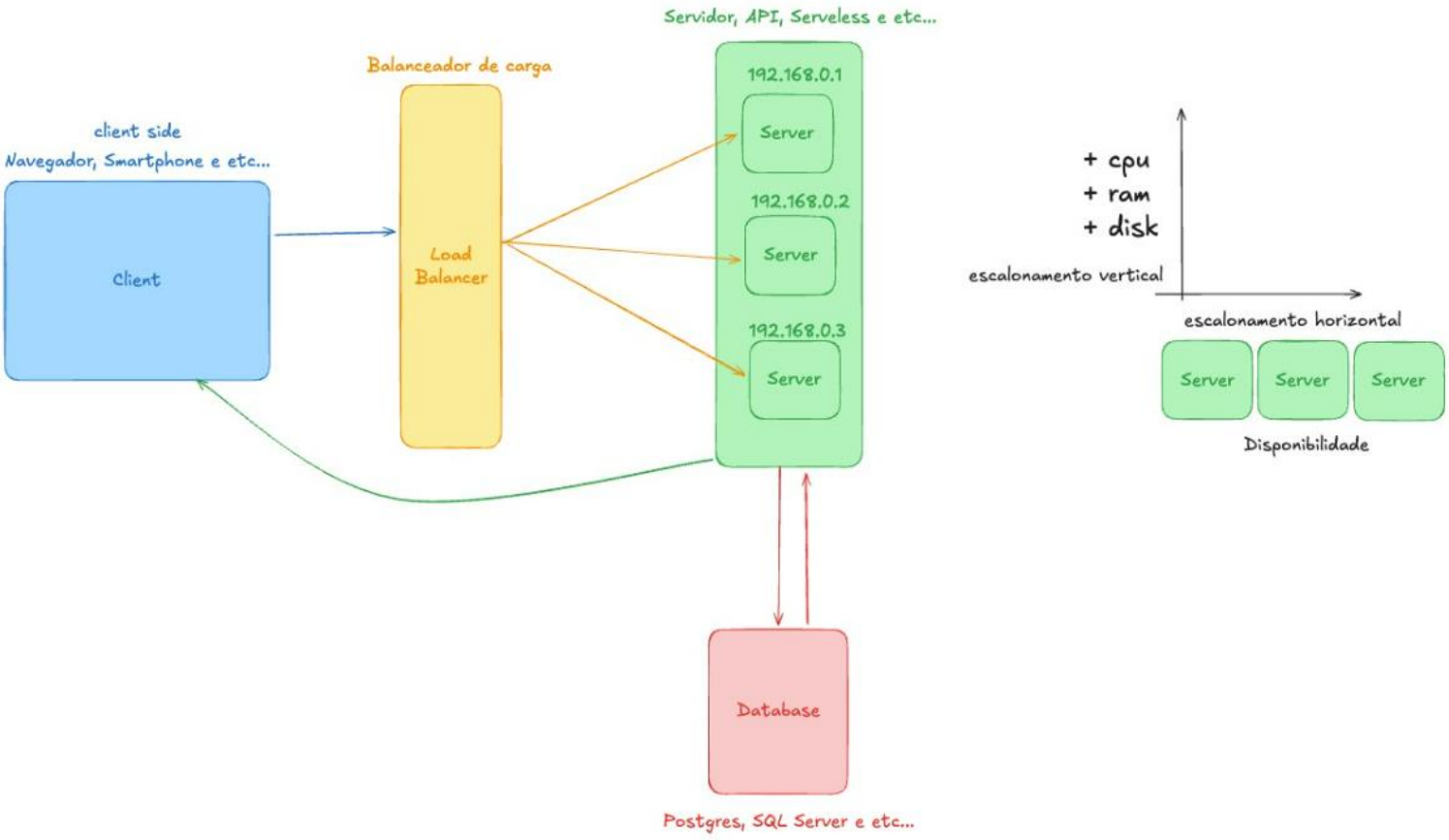
O projeto é totalmente modularizado, com scripts independentes para:

- orquestração
- execução
- comparação
- métrica

5. Algoritmos de Escalonamento

5.1 Round Robin (RR)

- utiliza um quantum configurado no JSON
- tarefas ficam rodando em ciclos
- prioridade é desconsiderada



5.2 Shortest Job First (SJF)

- ordena pelo menor tempo de execução
- ideal para reduzir tempo médio de espera
- Pode causar starvation (processos longos ficam esperando muito)

5.3 Prioridade

- executa tarefas de prioridade 1 primeiro
- empates - SJF interno (critério secundário)

6. Fluxo de Execução do Sistema

1. launcher.py lê config.json
2. Algoritmo desejado é selecionado
3. O conjunto de tarefas é carregado
4. As tarefas são ordenadas conforme o algoritmo
5. main.py simula o processamento com time.sleep()
6. Resultados são salvos em metricas.json
7. comparador.py pode analisar os resultados

7. Métricas Geradas

- tempo médio de resposta
- tempo total de execução
- ordem das tarefas
- algoritmo escolhido
- número de tarefas processadas

8. Conclusão

O desenvolvimento deste projeto possibilitou a criação de um sistema completo para a simulação de escalonamento de tarefas, utilizando exclusivamente a linguagem Python. A arquitetura modular, composta por scripts independentes para execução, análise e comparação, possibilitou uma organização clara e flexível, aproximando a solução de sistemas reais de orquestração de cargas de trabalho.

Ao implementar e testar algoritmos como Round Robin, Shortest Job First e Prioridades, tornou-se possível compreender de forma prática como cada política influencia diretamente o desempenho, o tempo de resposta e o comportamento das filas de tarefas. A simulação evidenciou que a escolha do algoritmo adequado depende não apenas do cenário, mas também da natureza das requisições e do grau de justiça ou eficiência desejado.

O uso de arquivos JSON para entrada e saída reforçou a padronização e facilitou a manipulação dos dados, enquanto o módulo de comparação permitiu a análise objetiva dos diferentes algoritmos, ampliando o valor educacional do projeto. Toda a execução foi conduzida de forma controlada através de scripts Python, o que tornou o processo repetível, auditável e simples de estender.

Dessa forma, o projeto cumpriu plenamente seu propósito pedagógico: demonstrou, por meio de simulação, como os conceitos de escalonamento aprendidos em Sistemas Operacionais se aplicam na prática. Além disso, forneceu uma base sólida para futuras melhorias, como introdução de novos algoritmos, interface gráfica, paralelismo real ou integração com filas de mensagens. O resultado final é um sistema funcional, bem estruturado e alinhado às demandas contemporâneas de processamento e organização de tarefas.