Centro Universitário de Brasília
Faculdade de Tecnologia
Sistemas Operacionais | 2025.2
Prof. Me. Michel Junio Ferreira Rosa

Projeto Prático – *BSB Compute*: Orquestração de Tarefas

Introdução

O avanço da Inteligência Artificial e do aprendizado profundo gerou uma nova demanda: gerenciar múltiplos processos de inferência em paralelo. Cada requisição feita a um modelo de IA (por exemplo, reconhecimento facial, análise de texto, classificação de imagens) consome uma parte significativa dos recursos computacionais disponíveis — CPU, GPU e memória.

Empresas que oferecem serviços de IA em nuvem, como a *BSB Compute*, precisam lidar com milhares de solicitações simultâneas, provenientes de diferentes clientes, cada uma com urgência e carga de trabalho distintas. Para evitar sobrecarga, a empresa desenvolve um sistema de orquestração, responsável por distribuir tarefas de forma equilibrada entre vários servidores de inferência.

O desafio é garantir máximo desempenho e justiça no uso dos recursos, aplicando políticas de escalonamento de processos e comunicação entre processos (IPC), fundamentos centrais de sistemas operacionais.

Descrição do Problema

A empresa *BSB* Compute opera um cluster de inferência distribuído composto por servidores independentes, cada um com capacidade distinta de processamento. O sistema recebe continuamente requisições de inferência (por exemplo, “analisar imagem”, “traduzir texto”, “classificar sentimento”), que precisam ser encaminhadas para os servidores disponíveis conforme:

- Capacidade de processamento (peso atribuído a cada servidor);
- Prioridade da requisição (tempo de resposta exigido);
- Carga atual (número de tarefas em execução);
- Política de escalonamento definida (Round Robin, Prioridade ou SJF).

Cada servidor é representado por um processo independente, e o orquestrador central é outro processo responsável por gerenciar a fila de requisições e a comunicação com os servidores.

A meta é maximizar o *throughput* (tarefas por segundo) e minimizar o tempo médio de resposta, simulando o comportamento de um escalonador real que precisa decidir, em frações de segundo, qual processo executar.

Objetivos do Projeto

1. Implementar um sistema de orquestração de processos concorrentes que distribui requisições de IA (*simuladas*) entre servidores de forma justa e eficiente.

2. Utilizar mecanismos de comunicação entre processos (IPC), como pipes ou filas de mensagens, para envio e retorno dos resultados.
3. Simular diferentes políticas de escalonamento (ROUND ROBIN, SJF, Prioridade).
4. Monitorar o desempenho do cluster, registrando tempos médios de resposta, uso de CPU e carga de cada processo.
5. Criar uma simulação dinâmica, com novas requisições chegando em tempo real (*temo de chegada aleatório*) e decisões automáticas de redistribuição.

Arquitetura do Sistema

O sistema é composto por dois níveis de processos:

1. **Orquestrador Central (*Master*)**
   - Responsável por receber requisições, ordená-las e distribuí-las conforme a política de escalonamento.
   - Mantém uma fila de requisições pendentes e monitora os estados dos servidores.
   - Comunica-se com os servidores por pipes, sockets ou filas de mensagens.
2. **Servidores de Inferência (*Workers*)**
   - Cada servidor representa um nó do cluster.
   - Executa tarefas simuladas (ex.: "classificação de imagem", "análise de texto").
   - Reporta ao orquestrador quando uma tarefa é concluída, liberando capacidade para novas inferências.

Especificações Técnicas

Entrada de Dados

- Número de servidores (**S**)
- Capacidade de cada servidor (**C_1, C_2, ..., C_n**)
- Número total de requisições (**N**)
- Prioridade de cada requisição (**1 = alta, 2 = média, 3 = baixa**)
- Tempo de execução estimado (**simulado em segundos**)

Exemplo de entrada (*simulada ou lida de arquivo JSON*):

```
{
  "servidores": [
    {"id": 1, "capacidade": 3},
    {"id": 2, "capacidade": 2},
    {"id": 3, "capacidade": 1}
  ],
  "requisicoes": [
    {"id": 101, "tipo": "visao_computacional", "prioridade": 1, "tempo_exec": 8},
    {"id": 102, "tipo": "nlp", "prioridade": 3, "tempo_exec": 3},
    {"id": 103, "tipo": "voz", "prioridade": 2, "tempo_exec": 5}
  ]
}
```

Políticas de Escalonamento Suportadas

1. **RR (Round Robin)**
2. **SJF (*Shortest Job First*)**
3. **Prioridade**

Saída Esperada

Durante a simulação, o sistema deve exibir logs em tempo real, como:

```
-----------------------------------------------
[00:01] Requisição 101 (Alta) atribuída ao Servidor 1
[00:03] Requisição 102 (Baixa) atribuída ao Servidor 3
[00:04] Servidor 1 concluiu Requisição 101
[00:06] Servidor 1 recebeu nova Requisição 104 (Média)
[00:07] Requisição 105 (Alta) redirecionada do Servidor 3 para o Servidor 2
-----------------------------------------------
Resumo Final:
Tempo médio de resposta: 6.2s
Utilização média da CPU: 84%
Taxa de espera máxima: 4.3s
Throughput: 0.97 tarefas/segundo
-----------------------------------------------
```

Implementação

Linguagem: C, Python ou Java.

Fluxo principal

1. Criar um processo principal (**orquestrador**).
2. Criar subprocessos (**servidores**).
3. Gerenciar uma fila de requisições e aplicar política de escalonamento.
4. Implementar migração de tarefas entre servidores sobrecarregados.
5. Utilizar pipes, sockets ou filas para comunicação.
6. Implementar logs automáticos de eventos e métricas.

Critérios de Avaliação

Critério	Descrição	Pontos
Modelagem de processos e IPC	Implementação correta de processos e comunicação entre eles.	3
Escalonamento e redistribuição	Funcionamento justo e eficiente das políticas de escalonamento.	3
Medições de desempenho	Registro claro de tempos, filas e uso de recursos.	2
Robustez e sincronização	Prevenção de deadlocks e travamentos.	1
Relatório técnico e apresentação	Clareza, justificativa, logs e análise comparativa de políticas de escalonamento.	1
Total		10

Entrega

- Grupo: até 3 alunos.
- Prazo: 02/12/2025.
- Implementação do projeto com toda a documentação e manual de utilização/compilação.
- Todos os arquivos do projeto devem estar no github/gitlab.
- ***Demonstração prática em laboratório com simulação em tempo real.***