(Deep) Perfect Scheduler?

1 Introduction

Scheduling problems are central to computing systems, where jobs (or tasks) must be assigned to limited resources over time. The goal is often to optimize metrics such as average waiting time, latency[1], or system throughput. The complexity of the scheduling problem varies depending on system assumptions, such as the number of servers, whether preemption is allowed, and whether job durations and arrival times are known in advance.

2 Problem Setting

We consider a single-server queue where jobs arrive over time and require a certain amount of service. Each job is characterized by:

- Arrival time a_i
- Service time (or length) s_i

The server can execute one job at a time. When multiple jobs are available, a scheduling decision must be made.

3 Variants of the Scheduling Problem

We categorize the problem variants by increasing complexity:

Setting	**Durations Known?**	**Arrival Times?**	**Preemption?**
SJF (offline)	Yes	All at time 0	No
Online SJF	Yes	Varying	No
SRPT	Yes	Varying	Yes
Unknown lengths	No	Varying	Possibly
Multiple servers	Mixed	Varying	Possibly

Table 1: Variants of the scheduling problem by system complexity.

[1] **Latency** refers to the total time a job spends in the system from its arrival to its completion. It includes both the waiting time and the service time. Minimizing latency is crucial in systems where responsiveness is key, such as in real-time processing or user-facing services.

4 Illustrative Example

Consider three jobs:

- Job T1: arrival at $t = 0$, service time = 10
- Job T2: arrival at $t = 1$, service time = 2
- Job T3: arrival at $t = 2$, service time = 1

Non-Preemptive SJF

The server starts T1 at $t = 0$ and runs it to completion (since it is first to arrive), followed by T2 and then T3. Resulting waiting times:

- T1: 0
- T2: 9
- T3: 10

Average waiting time = 6.33

Preemptive SRPT

The scheduler always runs the job with the shortest remaining time:

- Start T1 at $t = 0$
- At $t = 1$, T2 arrives (2s remaining) $\rightarrow$ preempt T1
- At $t = 2$, T3 arrives (1s) $\rightarrow$ preempt T2
- T3 completes at $t = 3$
- Resume T2 from $t = 3$ to $t = 5$
- Resume T1 from $t = 5$ to $t = 13$

Waiting times:

- T1: 3
- T2: 2
- T3: 0

Average waiting time = 1.67

5 Optimal Scheduling Policies

- **SJF (Shortest Job First)** is optimal for minimizing average waiting time when all jobs arrive at $t = 0$ and durations are known.
- **SRPT (Shortest Remaining Processing Time)** is optimal for minimizing total time in system when arrivals vary and preemption is allowed.
- If job durations are unknown, no deterministic scheduling rule can guarantee optimality.
- In multi-server systems, optimal policies are more complex and depend on load balancing considerations.

6 Real-World Scenarios for Scheduling

To illustrate scheduling problems in practical settings, we describe several real-life scenarios where task scheduling plays a critical role. These examples can be used to motivate simulation studies or benchmark heuristic scheduling strategies.

6.1 Web Server Request Scheduling

Incoming web requests arrive at unpredictable times and require variable processing time. A web server or cloud backend must decide which request to serve next.

- **Tasks:** HTTP requests
- **Processor:** Web server thread or cloud instance
- **Strategies:** FIFO, Shortest Estimated Time First
- **Data:** Web logs (e.g., NASA HTTP logs or simulated Poisson arrivals)

6.2 Print Queue Management

In an office setting, print jobs arrive with different page counts and processing times.

- **Tasks:** Print jobs
- **Processor:** Printer
- **Strategies:** FIFO, Shortest Job First
- **Data:** Simulated workloads or data from a university lab

6.3 Operating System Process Scheduling

Operating systems must schedule CPU time among multiple active processes, each with different workloads and possibly priorities.

- **Tasks:** OS processes
- **Processor:** CPU
- **Strategies:** Round-robin, SJF, SRPT
- **Data:** Linux trace logs or simulation with tools like SimSo

7 Challenges and Open Problems

- **Unknown job durations:** requires estimation, learning, or adaptive strategies
- **Scalability:** MDP state space grows rapidly with number of jobs
- **Fairness:** Shortest-job policies can starve long jobs unless modified
- **Multi-server scheduling:** balancing jobs while minimizing wait requires decentralized or coordinated policies
- **Stochastic arrivals and heavy-tail durations:** complicate prediction and control

8 MDP Formulation

The scheduling problem can be modeled as a Markov Decision Process (MDP):

- **States:** a tuple encoding the current time, the running job, and a queue of waiting jobs with their arrival and remaining times

- **Actions:** select which job to run (preemption allowed)
- **Transitions:** governed by job completion and new arrivals (e.g., from a Poisson process)
- **Rewards:** negative of total waiting time, or delay per job
- **Policy:** maps a state to a job selection action

The goal is to learn or compute a policy that minimizes expected cumulative cost (e.g., average waiting time).

As Bachelor Thesis Opportunities, one can opt to focus on:

- Simulating realistic arrival patterns and processing times
- Comparing scheduling strategies under varying system loads
- Benchmarking algorithms on real or synthetic traces
- Studying extensions such as fairness constraints, task deadlines, or dynamic priorities