Scheduling Problem with Random Arrivals

Cortinovis Nicola, Lucas Marta
June 29, 2025

Introduction
This report focuses on the task scheduling problem with random arrivals. Scheduling is central to computing systems, where jobs must be assigned to limited resources over time, impacting the performance of servers, cloud platforms, and real-time applications. In our setting, job arrivals follow a Poisson process and are dispatched to one of three processors, with service times drawn from an exponential distribution or a log normal distribution. We implement and compare four scheduling strategies—First-In-First-Out (FIFO), Shortest-Job-First (SJF), Last-In-First-Out (LIFO) and preemptive LIFO (P-LIFO),—evaluating them with varying metrics. Finally, we analyze each strategy's performance under varying load conditions. Our GitHub code implementation of this project is available at [1].

1 Preliminaries and specific objectives

Before presenting our approach to solving the scheduling problem, we introduce key concepts that will aid in understanding our methodology.

1.1 Exponential Distribution

We say that a random variable X is distributed Exponentially with rate λ,

$$X \sim \text{Exp}(\lambda)$$

if X has the probability density function (p.d.f.):

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0, \\ 0 & x < 0. \end{cases}$$

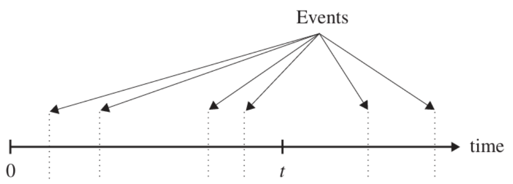
1.2 Lognormal Distribution

If $X \sim \mathcal{N}(\mu, \sigma)$, then the continuous variable $Y \sim e^X$ is said to have $Lognormal(\mu, \sigma)$ distribution. That is, Y is lognormal if $log(Y) = X \sim \mathcal{N}(\mu, \sigma)$. The p.d.f. for the log-normal is:

$$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(-\frac{(\ln x - \mu)^2}{2\sigma^2}\right), \quad x > 0$$

1.3 Poisson Process

Given a sequence of events we define $N(t)$ with $t \geq 0$ as the number of events that occurred by time t.



A Poisson process with rate λ is a sequence of events such that the inter arrival times are i.i.d. Exponential random variables with rate λ and $N(0) = 0$. [2]

1.4 Scheduling strategies

Scheduling is the process of deciding the order and timing in which tasks are executed. It is an extremely important topic in designing computer systems, since the right scheduling policy can vastly improve efficiency. Scheduling strategies are generally categorized into two primary types: preemptive and non-preemptive. Preemptive schedulers have the capability to interrupt a task in order to allocate resources to another task, whereas non-preemptive schedulers allow each task to run until it is fully completed. We employed the following scheduling strategies:

First In First Out (FIFO) A non-preemptive scheduling strategy that gives priority to jobs based on their arrival order, favoring those which arrive first.

Shortest Job First (SJF) A non-preemptive scheduling strategy that gives priority to jobs based on their length, favoring the shortest ones.

Last In First Out (LIFO) A non-preemptive scheduling strategy that gives priority to jobs based on their arrival order, favoring those which arrive last.

Preemptive Last In First Out (P-LIFO) A preemptive scheduling strategy that gives priority to jobs based on their arrival order, favoring those which arrive last and allowing them to interrupt ongoing jobs.

1.5 Project outline

In this project, we will examine the performance of various scheduling strategies as we alter the arrival rate of the Poisson process and vary the job duration distribution, considering both exponential and lognormal cases.

2 Methods

The problem we analyze involves modeling a task processing system where jobs of varying lengths arrive continuously over a fixed time period. The number of arrivals is modeled by a Poisson process with rate fixed to λ and the length of the processes can either follow a Exp(ρ) or a Lognormal(μ, σ). Our model has 3 identical servers that process jobs according to a specified strategy. Tasks are allocated to servers based on a shortest workload first policy, which takes into account both the remaining duration of the current job and the cumulative length of the jobs in the queue. To implement this model, we leveraged the SimPy library, a Python library for simulating scheduling and resource allocation in systems with competing, time-dependent tasks. [3]

2.1 Metrics

We monitored the subsequent metrics across all our experiments to collect data regarding the performance of various strategies from different perspectives:

- **Utilizations** (U_i): Fraction of time server i was busy during the simulation:

  $$\mathrm{U}_i = \frac{B_i}{T}$$

  where T is the total simulation time and B_i is how much a server was busy.

- **Throughput** (Th):

  $$\mathrm{Th} = \frac{J}{T}$$

  This represents the average rate of job completions per unit time.

- **Coefficient of Variation of Utilization** (CV_U):

  $$\mathrm{CV}_U = \frac{\sigma_U}{\mu_U}$$

  where μ_U is the mean utilization and σ_U is the standard deviation of utilizations. This metric reflects how balanced the load is across servers.

- **Mean Latency** (ML):

  $$\mathrm{ML} = \frac{1}{J}\sum_{j=1}^{J}(C_j - A_j)$$

  where A_j is the arrival time and C_j the completion time of job j. This metric captures the average time jobs spend in the system.

Finally, we also utilized a Gantt charts to plot how jobs are handled during the simulation by our servers. [4]

3 Results

To evaluate the performance of different scheduling strategies, we fixed a seed to allow reproducibility and fair comparison, then we conducted simulations varying the system load by testing two distinct arrival rates: a low rate of $\lambda = 1.5$ and a high rate of $\lambda = 3.5$. Each job's service time was drawn from one of two distributions: an exponential distribution Exp(1.0) with mean 1, and a log-normal distribution $\mathrm{LogNor}(0, 0.5)$. The log-normal parameters ($\mu = 0$, $\sigma = 0.5$) were carefully chosen to yield a mean close to that of the exponential distribution, while introducing greater variability to model burstier workloads. This allowed for a fair comparison between memory-less and heavy-tailed service time behaviors. For each setting, we simulated a system with 3 servers and recorded the aforementioned metrics. Each simulation ran for a fixed duration of 50 time units.

λ	Service Distr	ML	Th	U_i	CV_U
FIFO					
1.5	Exp(1.0)	1.51	1.16	[0.63, 0.48, 0.29]	0.30
1.5	$\mathrm{LogNor}(0, 0.5)$	1.36	0.80	[0.51, 0.24, 0.18]	0.47
3.5	Exp(1.0)	3.65	2.36	[0.82, 0.79, 0.73]	0.05
3.5	$\mathrm{LogNor}(0, 0.5)$	3.29	2.08	[0.92, 0.75, 0.78]	0.09
SJF					
1.5	Exp(1.0)	1.45	1.16	[0.63, 0.48, 0.29]	0.30
1.5	$\mathrm{LogNor}(0, 0.5)$	1.36	0.80	[0.51, 0.24, 0.18]	0.47
3.5	Exp(1.0)	2.40	2.60	[0.83, 0.79, 0.79]	0.02
3.5	$\mathrm{LogNor}(0, 0.5)$	2.46	2.16	[0.90, 0.80, 0.77]	0.07
LIFO					
1.5	Exp(1.0)	1.66	1.16	[0.63, 0.50, 0.26]	0.32
1.5	$\mathrm{LogNor}(0, 0.5)$	1.42	0.80	[0.51, 0.24, 0.18]	0.47
3.5	Exp(1.0)	2.39	1.96	[0.84, 0.84, 0.79]	0.03
3.5	$\mathrm{LogNor}(0, 0.5)$	2.43	2.08	[0.88, 0.80, 0.79]	0.05
Preemptive LIFO					
1.5	Exp(1.0)	1.62	1.16	[0.63, 0.45, 0.32]	0.28
1.5	$\mathrm{LogNor}(0, 0.5)$	1.64	0.80	[0.51, 0.24, 0.18]	0.47
3.5	Exp(1.0)	1.32	1.96	[0.81, 0.85, 0.78]	0.03
3.5	$\mathrm{LogNor}(0, 0.5)$	2.42	1.68	[0.90, 0.82, 0.79]	0.05

Table 1: Performance metrics for four scheduling strategies (FIFO, SJF, LIFO, Preemptive LIFO) under varying arrival rates and service time distributions.

We also plotted the Gantt charts for all the experiments, they can be viewed at the project GitHub repository. [1]

4 Discussion and conclusion

Our simulation findings underscore the trade-offs associated with various scheduling strategies in relation to differing load conditions and service time distributions. In scenarios characterized by low load, all strategies achieved equivalent Th, indicating the absence of queue buildup and, consequently, an underutilized system. This observation is corroborated by the consistently low and comparable U_i. Among the strategies evaluated, SJF demonstrated the lowest ML, thereby affirming its efficacy in prioritizing shorter tasks when capacity is available. Conversely, under high-load conditions, the observed results were more heterogeneous. The CV_U values across strategies experienced a notable decline, indicating a more equitable load distribution, which is further validated by the nearly saturated U_i values. Once again, SJF distinguished itself by achieving the highest Th while maintaining relatively low ML, attributable to its continuous prioritization of shorter jobs. P-LIFO exhibited a notable advantage in the Exp case, attaining the lowest ML by capitalizing on preemption, facilitating the immediate servicing of new short jobs without incurring penalties. However, this advantage dissipated in the LogNor case, where a higher probability mass is assigned to longer jobs. LIFO and FIFO strategies demonstrated more modest performance. LIFO maintained a relatively average ML and Th, whereas FIFO encountered the most significant challenges under load, exhibiting higher ML and mid-range Th. In conclusion, SJF is evidently the preferred strategy in low-load environments, while P-LIFO may offer benefits in high-load situations, particularly when new jobs have a high probability of being short.

References

[1] gitub.com/NicolaCortinovis/MAS_2025

[2] Harchol-Balter, M. (2013). Performance Modeling and Design of Computer Systems. Cambridge University Press

[3] SimPy library

[4] Clark, W., Polakov, W. N., & Trabold, F. W. (1922). The Gantt chart: A working tool of management. New York: The Ronald Press Company. Retrieved from https://archive.org