UNIVERSITY OF PISA

MSc in Computer Engineering

Project for Performance Evaluation of Computer Systems and Networks

Multi-core scheduling

Professors:

Prof. Giovanni Stea

Ing. Giovanni Nardini

Students:

Taulant Arapi (645308)

Francesco Barcherini (645413)

Antonio Ciociola (645324)

ACADEMIC YEAR 2024/2025

Contents

1 Introduction

1.1 Description and specification

It is required to analyze a multi-core computer equipped with N CPUs, which execute multiple interactive processes. The processes are dynamically generated at intervals of T seconds. Each process has a total duration D, which is divided into three distinct execution phases:

1. initial processing phase;
2. I/O operation phase;
3. final processing phase.

The times T and D are defined as IID random variables. The processes are categorized into two types and generated as *CPU bound* with probability p or *I/O bound* with probability $1 - p$. The difference is that:

- in CPU-bound processes the I/O operation phase constitutes 20% of the total duration D, and the other two phases both the 40%;
- in I/O-bound processes the I/O operation phase constitutes 80% of their total duration D, and the other two phases both the 10%;

Processes are assigned to CPUs by the operating system's scheduler, which selects tasks from a list of "ready" processes. When a CPU becomes idle for entering the I/O operation phase or ending the final processing phase of a process, the scheduler immediately assigns it to a new ready process, if present. Once a process completes its I/O operation phase, it is marked as ready again and is returned to the scheduling queue. A process exits the system once it has successfully completed all three phases of execution.

The scheduler can be implemented to follow two distinct policies:

- First Come First Served (FCFS): processes are scheduled in the order of their arrival after the generation or after the end of the I/O phase;
- Shortest Job First (SJF): scheduling is determined based on the shortest remaining time until either the process's I/O phase or its completion.

1.2 Objectives

The aim of this project is to analyze the execution of a multi-core scheduling system under varying operational scenarios.

One of the the objectives is to determine the optimal characteristics of the system in order to make it work without the need to discard processes. Furthermore, the analysis aims to assess the impact of the number of CPUs and of the distributions of generation and duration times as regards the system's performance. Another objective is to realize a resource efficiency evaluation.

In particular, the different configurations taken into consideration are the scheduling policies, the number of CPUs in the system, the type of the processes (CPU bound or I/O bound), the generation intervals and the duration of processes.

The overarching goal is to generate insights about the impact of the scheduling strategy and of the functional parameters of the system to evaluate the performance of multi-core environments, enhancing system efficiency and minimizing delays.

1.3 Indices

To evaluate the system's performance comprehensively, the following indices will be analyzed in detail:

- turnaround time R: defined as the total time elapsed from the arrival of a process in the system to its completion. This metric provides a measure of the computer's overall responsiveness and of the time required for different types of processes to be executed;
- waiting time W: the cumulative time a process spends in the ready queue before being assigned to a CPU for execution. This metric represents the impact on the turnaround time of the queuing policy and of the workload of the system;
- CPU utilization of the n_{th} CPU ρ_n: the fraction of time that each CPU is actively engaged in executing processes instead of being idle. This metric reflects the efficiency of resource usage within the system;
- active CPUs over time N_A: a dynamic measure of the number of CPUs actively processing tasks at any given moment. This index helps assess load distribution and system evolution;
- queue length over time N_q: tracks the size of the ready queue throughout the simulation, highlighting bottlenecks and variations in process scheduling.

These indices will be statistically analyzed to identify trends, anomalies, edge cases and key factors influencing system performance. By examining these metrics under diverse configurations of N, p and scheduler policies, the project aims to provide actionable recommendations for improving the efficiency and adaptability of multi-core scheduling systems.

1.4 Assumptions

To perform the analysis, the following assumptions are made:

- there is sufficient memory to store the list of processes ready for execution, so the ready queue will be considered as infinite and there is no need to discard processes;
- there is no overhead nor delay for every process from the generation to the arrival in the ready queue;
- there is no overhead nor delay for every process in the transitions between the ready queue, the CPUs and the I/O phase;
- the only workload of the CPUs is the execution of the processes. Scheduling, I/O, process generation do not require CPU time;
- the overhead for the sorting algorithm in SJF policy and for the queue handling is negligible;
- with the shortest job first (SJF) scheduling, an accurate estimate of the execution times is known a priori. If the scheduler is FCFS, this data is not used;
- the generation time, the duration time and the characteristic of a process of being CPU-bound or I/O-bound are respectively IID Random Variables;
- I/O operations and CPU processing do not interfere with each other.

1.5 Preliminary stability estimation

The system can be modeled as a queueing network as shown in Figure 1.1.

Let's analyze the system when the process generation time T and the process duration D are exponentially distributed, with mean values $E[T]$ and $E[D]$ respectively, and when the queues follow a FCFS scheduling policy. New processes are generated at rate $\gamma_1 = \frac{1}{E[T]}$. The initial processing phase and the final processing phase are modeled by the $M/M/C$ Service Center 1, where $C = N$ is the number of CPUs. Here it is assumed that the service time t_{cpu} can be $0.4D$ for CPU bound jobs and $0.1D$ for I/O bound jobs. This means that, for total probability theorem, the mean value of the service time of the CPU processing is

$$E[t_{cpu}] = 0.4E[D] \cdot p + 0.1E[D] \cdot (1-p) = 0.3E[D] \cdot p + 0.1E[D]$$

so that

$$\mu_{cpu} = \frac{1}{E[t_{cpu}]} = \frac{1}{0.3E[D] \cdot p + 0.1E[D]}$$

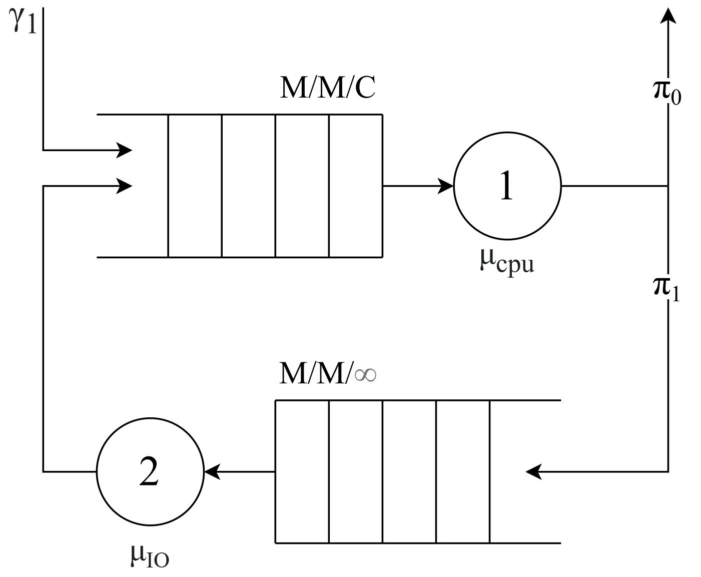


Figure 1.1: Model of the system as a queueing network

After the CPU processing phase, we can assume that it is equally likely that the process is at the beginning of the I/O phase or at the end of final processing phase. In the second case, it leaves the system with routing probability $\pi_0 = \frac{1}{2}$. If the process starts the I/O phase, it is routed with probability $\pi_1 = \frac{1}{2}$ to the $M/M/\infty$ Delay Center 2 with service time t_{IO}. Reasoning as before, the mean value of the service time of the I/O processing is $0.2D$ for CPU bound jobs and $0.8D$ for I/O bound jobs. The mean value of the service time of the I/O processing phase is

$$E[t_{IO}] = 0.2E[D] \cdot p + 0.8E[D] \cdot (1-p) = 0.8E[D] - 0.6E[D] \cdot p$$

so that

$$\mu_{IO} = \frac{1}{E[t_{IO}]} = \frac{1}{0.8E[D] - 0.6E[D] \cdot p}$$

Both service centers are of type $M/M/C$, the external arrival γ_1 is Poissonian, routing probabilities are state-independent and arcs are traversed in zero time. Therefore the hypothesis of Jackson's theorem are satisfied and the system admits a product form if the stability conditions $\rho_i = \frac{\lambda_i}{C_i \cdot \mu_i} < 1$ are satisfied for each service center i. To compute λ_1 and λ_2 we can define the routing matrix $\Pi = \begin{bmatrix} 0 & \frac{1}{2} \\ 1 & 0 \end{bmatrix}$. Arrivals are $\gamma = \begin{bmatrix} \gamma_1 \\ 0 \end{bmatrix}$. It is $\lambda = (I - \Pi^T)^{-1} \cdot \gamma$. We can observe that, since the input and output must balance, $\gamma_1 = \frac{\lambda_1}{2}$ and from the routing we get $\lambda_2 = \frac{\lambda_1}{2}$. At the end we get $\lambda_2 = \gamma_1 = \frac{1}{E[T]}$ and $\lambda_1 = \frac{2}{E[T]}$.

The stability condition for the delay center is always true, so the system is stable if the first service center is stable. To have $\rho_1 < 1$ we get $\frac{\lambda_1}{N \cdot \mu_{cpu}} < 1$: substituting and rearranging we get

$$N > \frac{3p+1}{5} \cdot \frac{E[D]}{E[T]} \tag{1.1}$$

The metrics to be computed can be obtained from the analysis of the two service centers. The mean number of busy CPUs is, for example,

$$E[N_{cpu}] = \frac{\lambda_1}{\mu_{cpu}} = \frac{3p+1}{5} \cdot \frac{E[D]}{E[T]} \tag{1.2}$$

And, with some computation based on known formulas, it is possible to get a theoretical estimation of the mean number of processes in the ready queue $E[N_q] = E[N_{q_1}]$, of the mean waiting time in the ready queue $E[W] = 2E[W_1]$ and of the turnaround time $E[R] = E[W] + E[D]$.

2 Implementation

We implemented the Computer system in OMNeT++.
Figure Figure 2.1 provides an overview of the system implementation. Below is a detailed explanation of each module:

- `ProcessGenerator`: this module generates processes according to a specified interarrival time distribution. Each process is assigned a total duration D, and the duration of each phase (initial, I/O, and final) is set to $0.4D$, $0.2D$, and $0.4D$ if the process is CPU bound, or $0.1D$, $0.8D$, and $0.1D$ if the process is I/O bound. The process is then sent to the scheduler.
- `scheduler`: this module contains an infinite-capacity queue. If **`isFCFS`** is set to **`true`**, a FIFO queue is used. Otherwise, a priority queue is used, with processes ordered by increasing duration until the I/O phase or termination (shortest job first).
  When a new process arrives, the scheduler sends it to the first free CPU. If no CPU is free, the process is enqueued. When a CPU is freed, the next process in the queue is sent.
  When a process enters the I/O phase, the scheduler waits for the specified I/O phase duration and adds the process back into the queue.
  When a process terminates, its turnaround time and waiting time are emitted as signals.
  The scheduler also keeps track of the number of active CPUs and the queue length, emitting those every time a message is received.
- `cpu`: There are N CPUs which operate independently. Each one, when it is free, receives a process from the scheduler and simulates its execution. When it finishes the execution phase (i.e. the I/O phase is reached or the process terminates), a message is sent to the scheduler, indicating that the CPU is now free. Every CPU records its utilization, emitting the busy time as a signal when the CPU is freed.

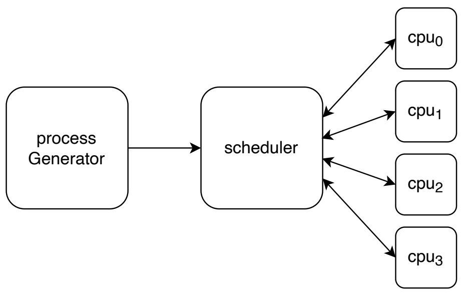


Figure 2.1: Overview of the system implementation.

3 Verification

3.1 Code verification

The code has been verified to work as expected with the debug GUI that comes with OMNeT++. In particular, the code has been tested trying to execute all the possible paths of the code in order to simulate all different cases, and comparing the actual state of the system with the expected one.

For the analysis, the specific constructs of OMNeT++ (such as cQueue for the queues or user defined messages) allowed to better follow the flow of events in the simulations. In fact the inspection of the content of messages and the state of the queues has been crucial to understand the behaviour of the system.

Additional information has been gathered printing log messages in the console with the use of `EV`, in order to have a better comprehension of the flow of the simulation. For example, `EV` has been very useful to track the main events of the simulation, such as the arrival of a new process, the start and end of the three phases of execution, the assignment of a process to a CPU. A simple graphical representation of the processes queued waiting to be assigned to a CPU helped to verify the correct behaviour of the scheduler in both FCFS and SJF policies.

3.2 Degeneracy test

Degeneracy tests aim to verify the correct behaviour of the system at extreme values of the parameters. The main quantitative parameters are the number of CPUs `N`, the probability of a process to be CPU bound `p`, the mean generation time `E[T]` and the mean process duration `E[D]`. The possible scenarios can be tested with FCFS or SJF scheduling policy and with exponential or uniform distributions of the two times.

The probability `p` set at 0 or 1 leads to the generation of only I/O bound or CPU bound processes. The simulation works as expected in the two cases with different values of the statistics (ex. turnaround and waiting time, number of busy processors etc.), without affecting the functionality of the system.

When the number of CPUs is set to 0, no CPU is instantiated and the processes are never assigned to a CPU. The system works as expected, with the processes waiting in the queue until the finish of the simulation time or the end of memory availability. Tests with only 1 CPU show a good behaviour and no problem related to the handling of vectors of gates with length 1. The test with a huge number of CPUs (100) is not different from every test with more than one CPU with respect to the multicore functionalities of the computer; obviously, in this case, there is always an available CPU to assign a process to and there is no queueing.

Talking about the mean generation time `E[T]` and the mean process duration `E[D]`, what matters is the ratio between the two. To simulate scenarios with very large (small) generation time and very small (large) process duration, the two parameters have been set to 10^3ms and 10^{-3}ms respectively, and vice versa. In the case of `E[T]` >> `E[D]`, every process executes its three phases before the generation of a new one, so there is no queueing as expected. In the case of `E[T]` << `E[D]`, the processes are generated at a very high rate so it is necessary to set `sim-time-limit` to a small value to end the simulation in a reasonable time. The system works as expected, with a high number of processes in the queue and CPUs always busy.

3.3 Consistency test

The consistency test aims to observe a similar behaviour of the system when the parameters are changed in such a way that the result is expected to be still the same.

With the hypothesis of First Come First Served scheduling policy and exponential distributions of times, the first tests verify the scenarios with fixed `p` = 0.5 and `N` = 6 and changing values of `E[T]` and `E[D]`, keeping the ratio $\frac{\texttt{E[D]}}{\texttt{E[T]}}$ still constant and equal to 10. In this way, the generation rate and the service rate change in the same way and the statistics related to the utilization should remain the same.In particular, the values chosen are 1ms, 4ms and 10ms for `E[T]` and 10ms, 40ms and 150ms for `E[D]`. The simulation effectively shows the same CPU utilization around the value of 0.835. Also the mean number of busy CPUs is constant and equal to 5 and the mean number of processes in the ready queue is around 4.7. The mean turnaround time, waiting time in the queue and service time show, instead, a proportionality with `E[D]` $\propto \frac{1}{\mu}$ as expected. The times are multiplied with a factor 4 and 10 in the second and third scenarios with respect to the first one.

A second consistency test consists in changing also the number of CPUs `N`, changing the ratio $\frac{\texttt{E[D]}}{\texttt{E[T]}}$ but keeping constant $\frac{\texttt{E[D]}}{\texttt{E[T]}\cdot\texttt{N}}$, proportional to the utilization of the CPUs $\rho = \frac{\lambda}{N \cdot \mu}$. The chosen values are 3, 6 and 9 for `N`, 1ms, 4ms and 10ms for `E[T]` and 5ms, 40ms and 100ms for `E[D]`. The CPU utilization extracted from the simulation effectively remains the same in the different scenarios and equal to 0.835. The mean number of busy CPUs increases proportionally with $\frac{\texttt{E[D]}}{\texttt{E[T]}}$: 2.5, 5 and 7.5 in the three cases. The other metrics change without a clear proportionality with the parameters, still consistently with the changes and with the expectations for a M/M/C system.

Another important test consists in changing these parameters in conditions of instability. For example, in the latter case with `N` equal to 1, 2 and 3 the system keeps being unstable: the CPUs cannot handle the load of the processes and the queue grows indefinitely.

The same tests have been performed combining the scenarios with the Shortest Job First queueing policy and with uniform distributions of the times. The results confirm the consistency of the system with the expected behaviour in term of CPU utilization and stability conditions. Of course there is no more the same proportionality between the times and the metrics as regards queueing, while it is still valid for the service times of the processes.

3.4 Continuity test

The continuity test verifies that a slight change in the input does not cause wild changes on the system behaviour.

In the system, the test is handled in both FCFS and SJF scheduling policies. The parameters vary with small steps of 5% of their value, so that `N` takes values 19, 20 and 21, `p` takes values 0.475, 0.5 and 0.525, `E[T]` takes values 1.425ms, 1.5ms and 1.575ms and `E[D]` takes values 47.5ms, 50ms and 52.5ms. The system is stable in all the cases, as expected, and the variation of the metrics is around no more than 10% of their value. For example, the CPU utilization varies from a minimum of 0.817 to a maximum of 0.854 (4.33% of variation). The turnaround time goes from 49.7ms to 55.4ms (10.3%), and similarly do the other statistics.

4 Analysis

4.1 Calibration of the parameters

Having a lot of parameters to calibrate, we decided to limit the number of possible values for each parameter. This choice was made to reduce the number of simulations to be performed and to have a more manageable number of results to analyze. The parameters that we decided to calibrate are the following:

- `pCpuBound`: $\{0.1, 0.9\}$
  The probability of a process being CPU-bound has two values, one for a system with a high number of I-O bound processes and the other for a system with a high number of CPU-bound processes.
- `meanProcessDuration`: $\{0.1\,\mathrm{s}\}$
  We chose a fixed value for the mean process duration, by doing this we can focus on the effect of the other parameters on the metrics.
- `meanGenerationTime`:
  The mean generation time was chosen to have a comparable load on the system for both values of `pCpuBound`. For each value of `pCpuBound` there is a simulation with heavy load and one with a lighter load. The values are in the order of tens of milliseconds.
- `numCpus`: $\{4, 12\}$
  The number of CPUs was chosen to simulate quad-core, and newer systems with 12 cores.
- `isFCFS`: {true, false}
  The scheduling policy that the scheduler uses can be either First-Come-First-Served or Shortest-Job-First.
- `generationType`: {"exponential", "uniform"}
  In addition to the exponential distribution, we also considered the uniform distribution for the generation of processes.
- `durationType`: {"exponential", "uniform"}
  In addition to the exponential distribution, we also considered the uniform distribution for the duration of the processes.

4.2 Time parameters setup

4.2.1 Warm-up period

To determine the warm-up period, we observed the evolution of the mean number of busy CPUs over time, through 10 independent runs and with different parameters configurations. The number of busy CPUs is a good indicator of the system's stability, as it doesn't depend on the scheduling policy. Our observations showed that the system stabilizes well before 200 s in all tested configurations. To handle variability and ensure robustness in worse cases, we set the warm-up period to 200 s.

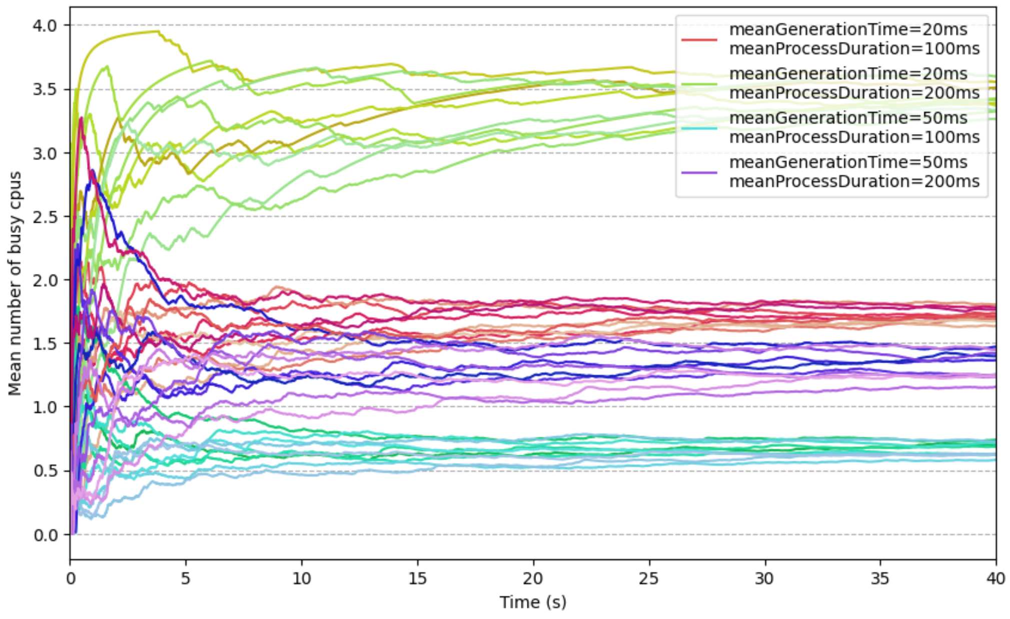


Figure 4.1: Line chart of the mean number of busy CPUs over time for different configurations of meanGenerationTime and meanProcessDuration and multiple repetitions.

For a better analysis, in the Figure 4.2 we show the average and standard deviation of the mean number of busy CPUs over the repetitions. Effectively, the mean and the standard deviation stabilize after 10^2s in all the scenarios.

The numerical analysis is reported in Table 4.1. Observing the evolution of the mean number of busy CPUs over time and of the standard deviation, with particular focus on the stabilization of these values in the last rows of the table, we can finally conclude that the warm-up period can be set to 200 s.

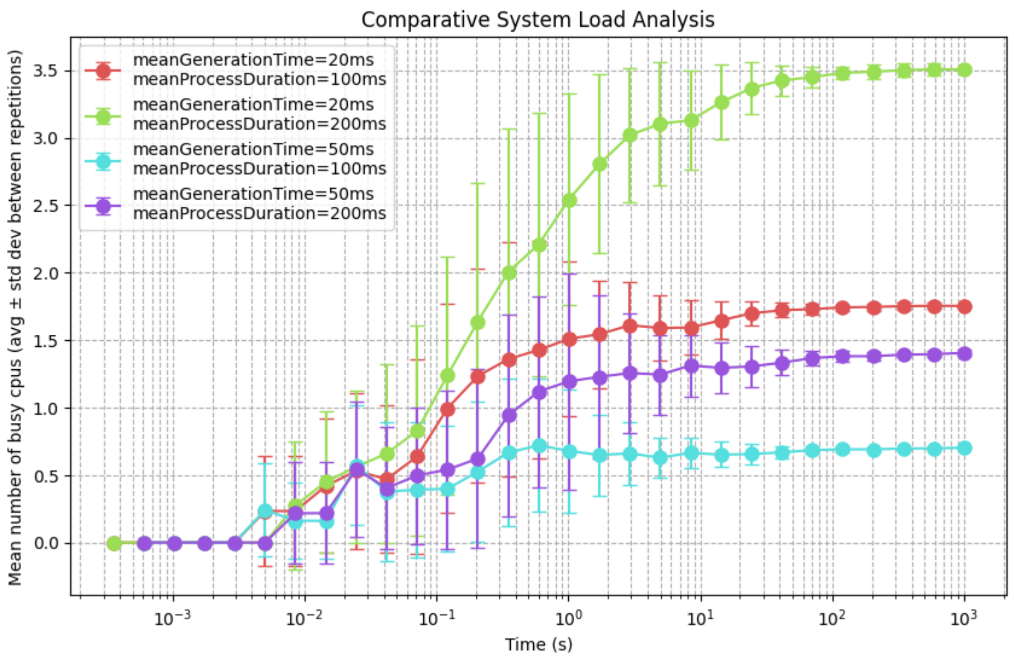


Figure 4.2: Average and Std dev of the mean number of busy CPUs over the repetitions.

Time (s)	20ms, 100ms		20ms, 200ms		50ms, 100ms		50ms, 200ms	
	Avg	Std Dev	Avg	Std Dev	Avg	Std Dev	Avg	Std Dev
1	1.52	0.58	2.54	0.78	0.68	0.46	1.20	0.80
2	1.54	0.37	2.85	0.64	0.67	0.27	1.25	0.51
5	1.59	0.24	3.10	0.46	0.63	0.15	1.24	0.29
10	1.63	0.18	3.18	0.34	0.65	0.10	1.29	0.20
20	1.68	0.11	3.33	0.22	0.64	0.08	1.27	0.15
50	1.72	0.05	3.43	0.10	0.68	0.05	1.35	0.09
100	1.74	0.02	3.47	0.05	0.69	0.02	1.37	0.04
200	1.75	0.02	3.49	0.05	0.69	0.02	1.38	0.03
500	1.75	0.03	3.50	0.05	0.70	0.01	1.40	0.02
1000	1.75	0.01	3.51	0.02	0.70	0.01	1.41	0.02

Table 4.1: Average and Std dev of the mean number of busy CPUs over the repetitions.

4.2.2 Simulation duration

For the simulation duration, we chose to end it at 1000 s. This ensures that after discarding the initial 200 s warm-up period, there remains 800 s of simulation data for analysis. Given that the mean generation time is on the order of tenths of a second, this duration strikes a balance between obtaining statistically meaningful results even after subsampling and maintaining reasonable simulation times.

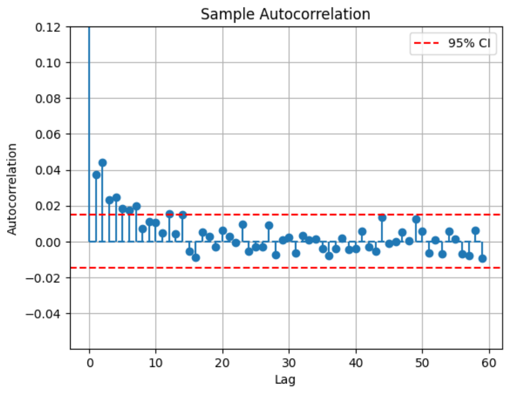
4.3 Subsampling

To analyze the data, the assumption of IID-ness will be needed, but without further modification the samples do not uphold it. For instance, if a process finishes with a large turnaround time, which is caused by a long queue, it is likely that the same thing will happen for the next processes.

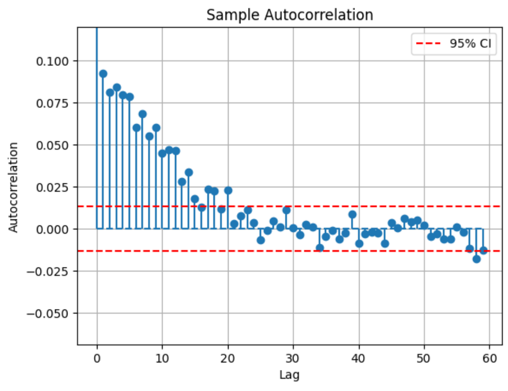
To address this and ensure independence between samples, subsampling has been employed. The new sample is constructed taking each point of the starting sample with probability p. $p = \frac{1}{2^k}$ and k is the smallest integer that passes the Ljung-Box test. Using the Ljung-Box test makes it possible to automate the subsampling step. It was implemented to test the first 30 lags, with a significance level of 5%, with these values the sample is considered independent if:

$$Q = n(n+2)\sum_{k=1}^{30}\frac{\hat{\rho}_k^2}{n-k} < \chi^2_{0.05,30} \approx 43.77 \tag{4.1}$$

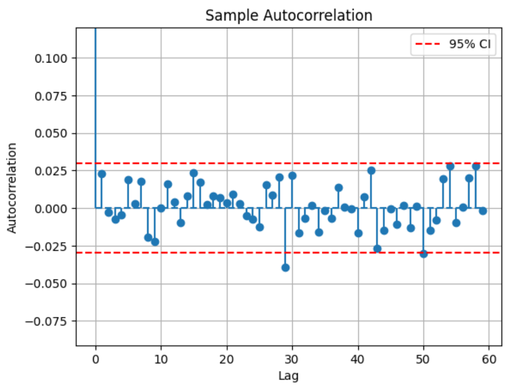
As utilization increases, the correlation becomes stronger, as shown in fig. 4.3.



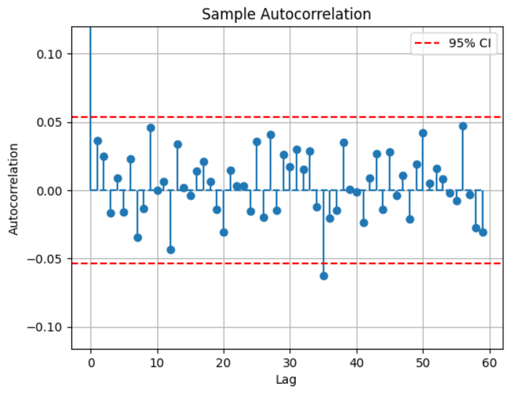
(a) $\rho = 0.4$, Q = 119



(b) $\rho = 0.5$, Q = 1333



(c) $\rho = 0.4$, $p = 1/4$, Q = 27



(d) $\rho = 0.5$, $p = 1/16$, Q = 22

Figure 4.3: Comparison of turnaround time autocorrelation with and without subsampling for different loads.

4.4 Statistical analysis

4.4.1 First Come First Served

For first, we analyze the First Come First Served (FCFS) scheduling policy. In the next section we will analyze the Shortest Job First (SJF) policy and compare the two scenarios.

Turnaround time

The first metric to be analyzed is the turnaround time. From fig. 4.4 and fig. 4.5 we can see that a larger utilization of the CPUs ρ skews the distribution of turnaround time to the right. This can be explained by the fact that the more the CPUs are utilized, the more probable is for a process to be queued and to wait more. The effects of ρ are more pronounced with fewer CPUs (as expected), and a higher `pCpuBound`. In fact, with a higher `pCpuBound` a process spends most of its time in the CPU, increasing the probability of forming a queue of ready processes.

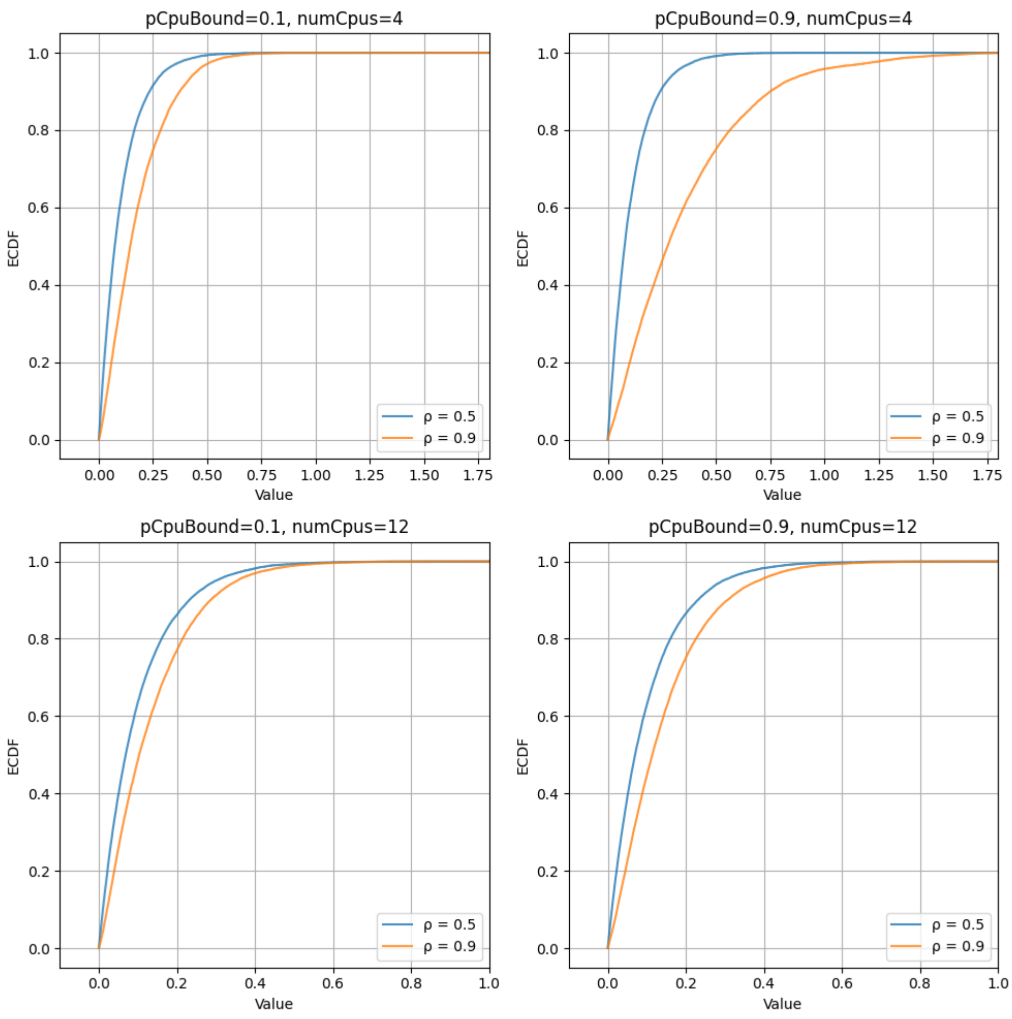


Figure 4.4: Empirical CDF of the turnaround time of the systems.

The empirical probability density function of the turnaround time, shown in fig. 4.5, confirms the previous observations. Comparing the plot above to the right with the others, we can observe that the distribution is more uniform and skewed to the right when ρ is higher, the processes are more likely CPU bound and there are few CPUs. In this case, the turnaround time can be modeled as the sum of the waiting time in the queue with the service time of the process.

When, instead, the utilization and the `pCpuBound` are lower and the number of CPUs is higher, the distribution is approximately an exponential: in this case the effects of queueing are less pronounced and the turnaround time is mainly determined by the (exponential) service time of the process.

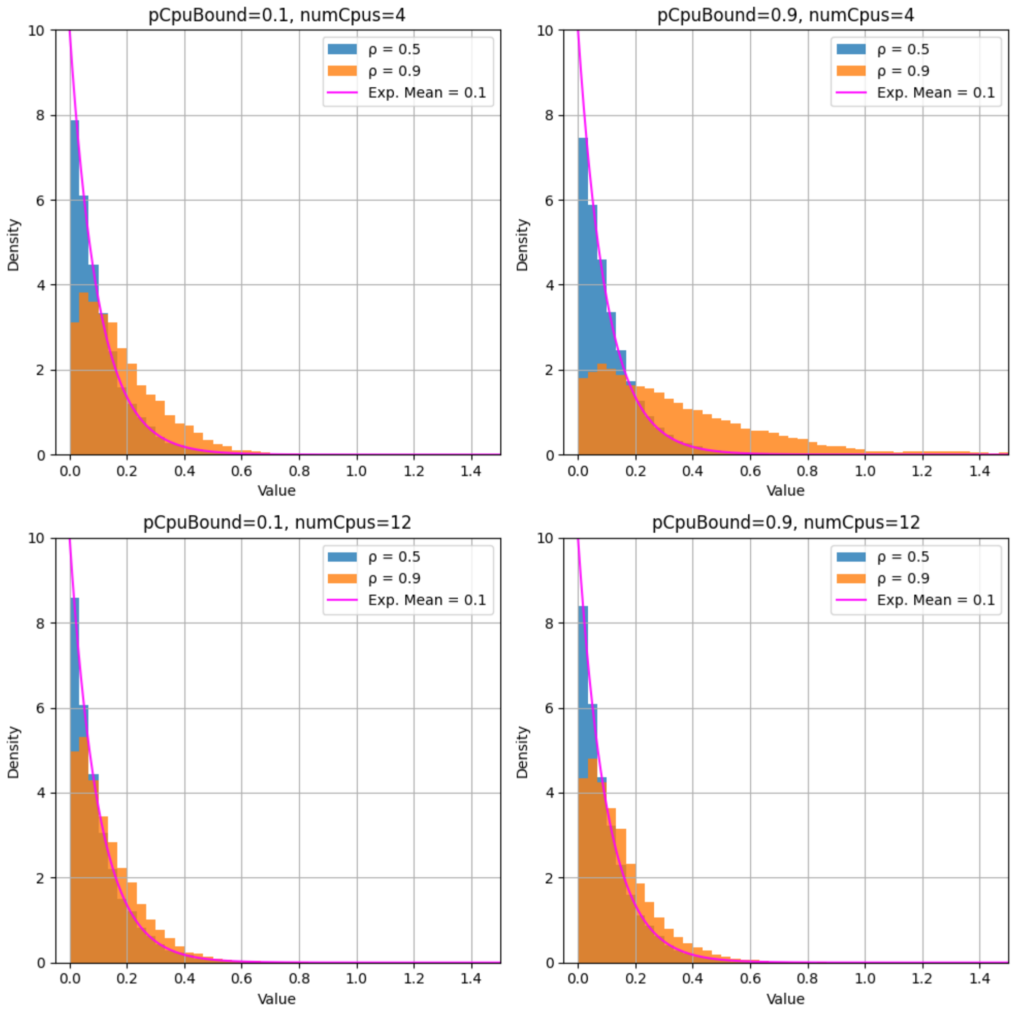


Figure 4.5: Density plot of the turnaround time of the systems.

From the Lorenz curve in fig. 4.6 we can see that the distribution that the turnaround time assumes with bigger ρ is more fair than the one with smaller ρ. We can explain this by the fact that when the system is more utilized, the processes are more likely to be queued and to wait a similar amount of time, making the distribution more fair.

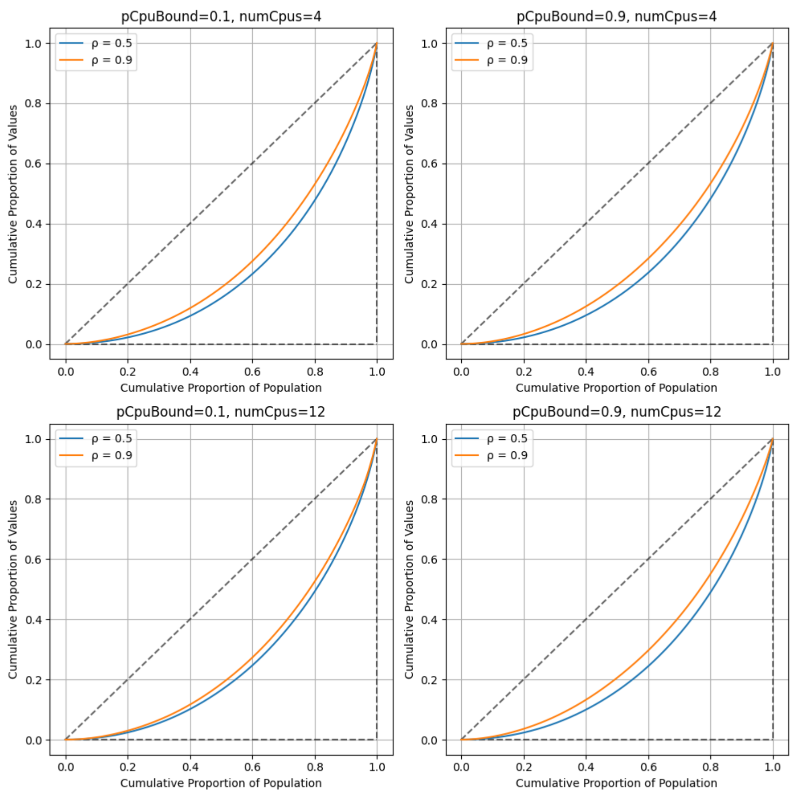


Figure 4.6: Lorenz curve of the turnaround time of the systems.

In fig. 4.7 we see that larger ρ values make the distribution more correlated, while more CPUs and a lower `pCpuBound` make the distribution less correlated.

This fits perfectly with the model: when the system is more utilized, the processes are more likely to be queued, so the waiting time of each process will be closely related to the one of its nearest processes in the queue, making the distribution more correlated. When the system is less utilized, the processes are more likely to be served immediately, making the distribution dependent only on the service time of the process.

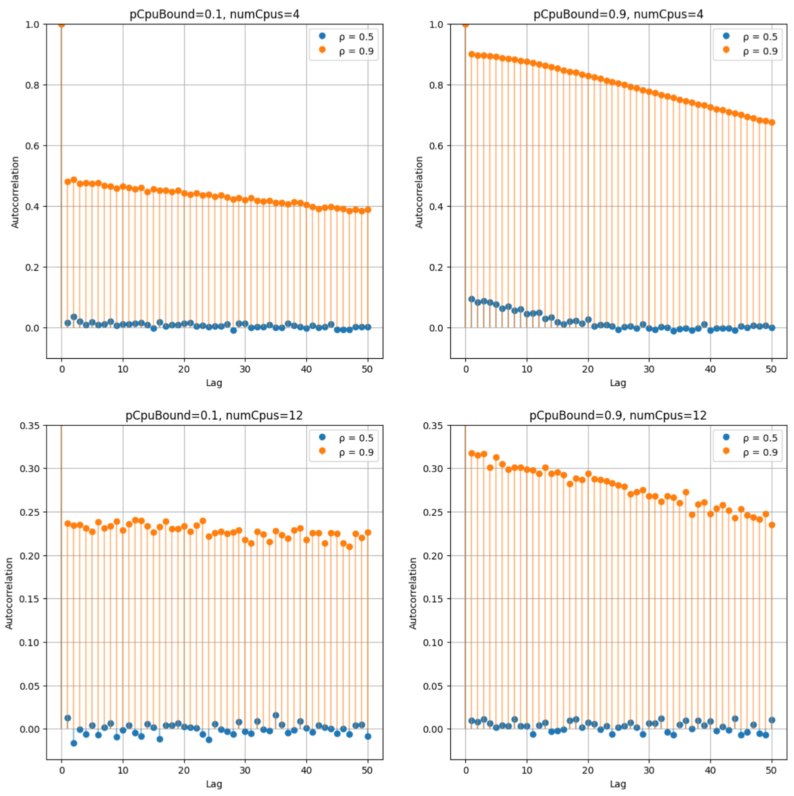


Figure 4.7: Autocorrelation plot of the turnaround time of the systems.

After using subsampling to make the data uncorrelated, we used bootstrap to calculate the confidence intervals of the mean and the standard deviation of the turnaround time. The `std dev` was calculated since we expect it to be equal to the mean when the distribution is exponential. It can be seen in table 4.2 that for small ρ they are similar.

		numCpus = 4				numCpus = 12			
		pCpuBound = 0.1		pCpuBound = 0.9		pCpuBound = 0.1		pCpuBound = 0.9	
Index		$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$
Mean	Value	103.1	195.0	108.5	358.4	99.6	132.5	100.1	144.6
	95% CI Low	101.2	181.2	102.3	321.2	97.7	121.5	98.6	131.4
	95% CI High	105.3	211.5	114.5	395.7	101.6	146.0	101.4	158.2
Std Dev	Value	101.3	137.2	108.1	316.6	99.3	113.5	98.8	126.7
	95% CI Low	98.6	126.2	102.2	280.2	96.7	102.4	97.0	111.8
	95% CI High	104.1	152.1	115.1	365.0	102.0	127.6	100.6	145.3

Table 4.2: Bootstrap results for turnaround time mean and Std Dev. (ms)

Waiting time

From fig. 4.8 and fig. 4.9 it is clear to see that for $\rho = 0.5$ the waiting time is almost always 0. For this reason in fig. 4.10 we see that the waiting time is more unfair when ρ is lower, since this is the behaviour of the line of maximum unfairness (all values equal to 0 except for one). In fig. 4.11 we see that even when $\rho = 0.5$ the times are really correlated.

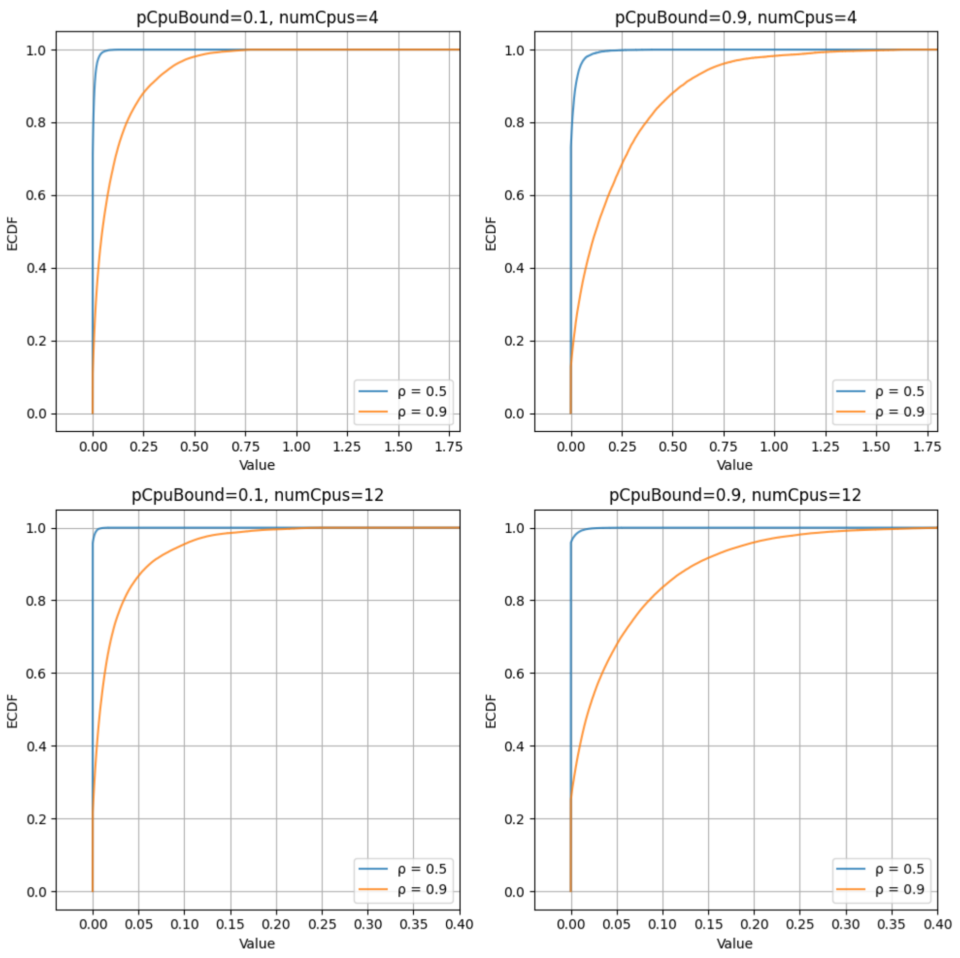


Figure 4.8: Empirical CDF of the waiting time of the systems.

From the plots of the empirical CDF it is possible to observe that the utilization of the CPUs impacts more than the variation of the single parameters `pCpuBound` and `numCpus` on the waiting time. Anyway, looking from left to right in particular at the impact of the first parameter, the time of CPU processing increases and the distribution moves to the higher values.

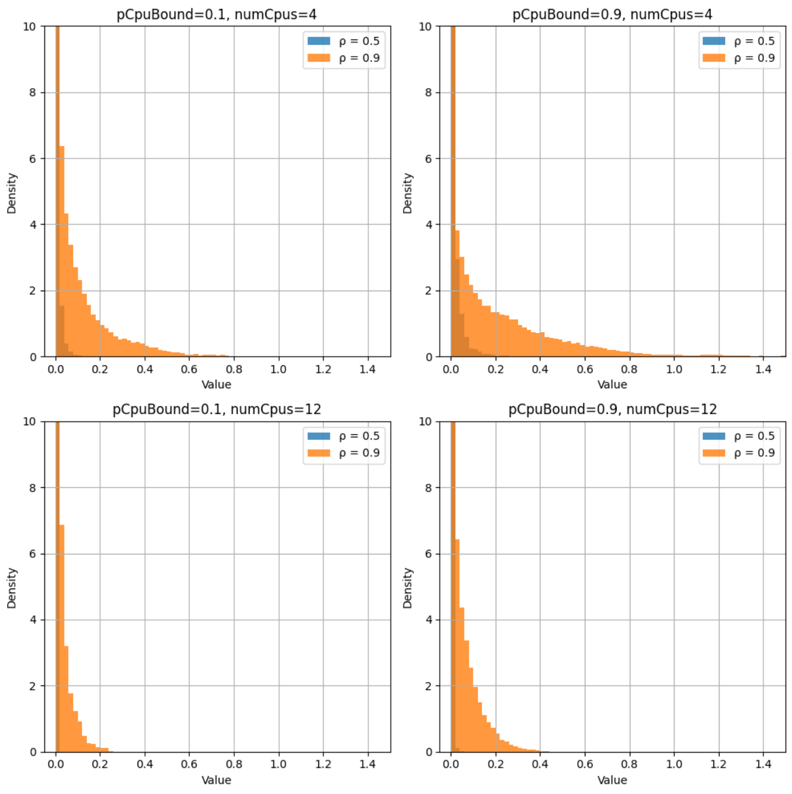


Figure 4.9: Density plot of the waiting time of the systems.

The analysis of the probability density function of the waiting time confirms that at low values of ρ the process are almost never queued. From left below to right above, the distribution is more uniform and skewed to the right, with the density concentrated less around the value 0.

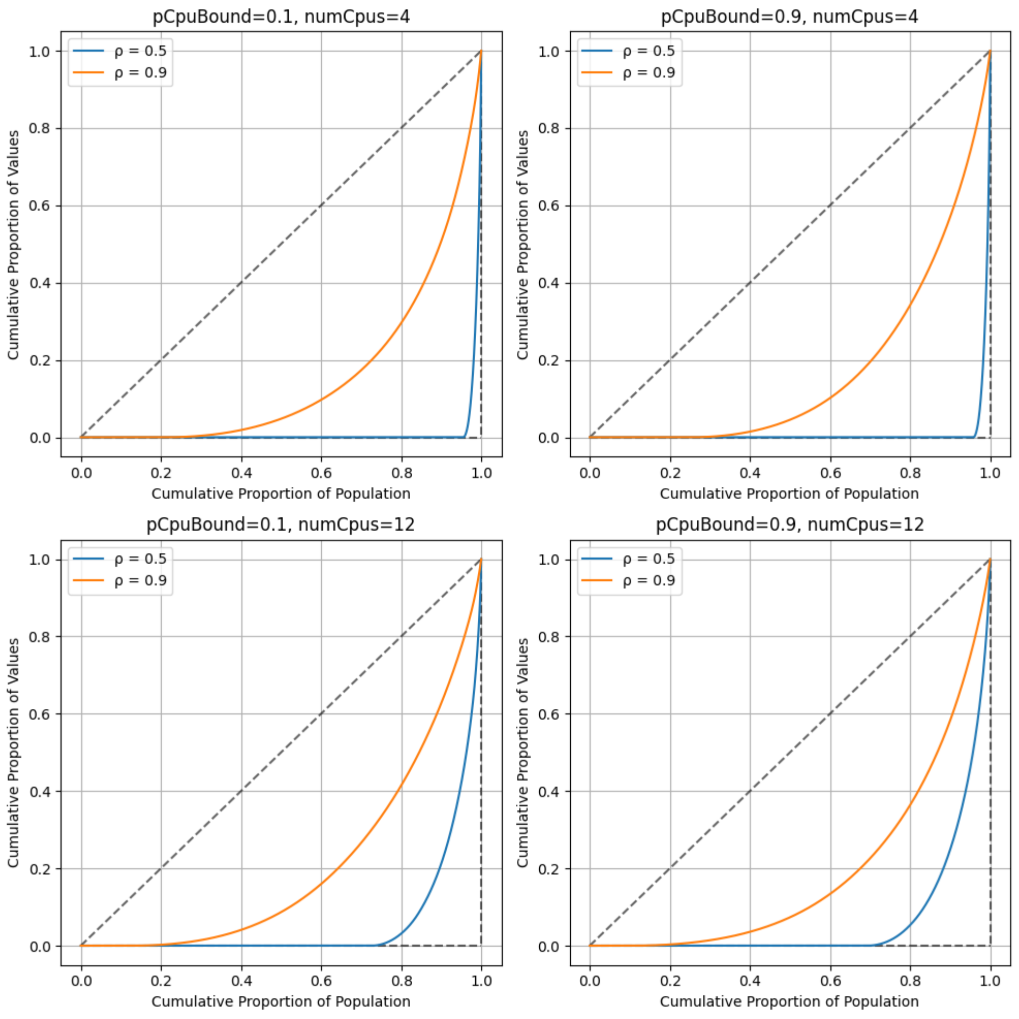


Figure 4.10: Lorenz curve of the waiting time of the systems.

We also analyzed the unfairness of the distribution of the wating time in the different scenarios, plotting the Lorenz curves in fig. 4.10. In every case the distribution is more unfair when ρ is lower, since the waiting time is almost always 0.

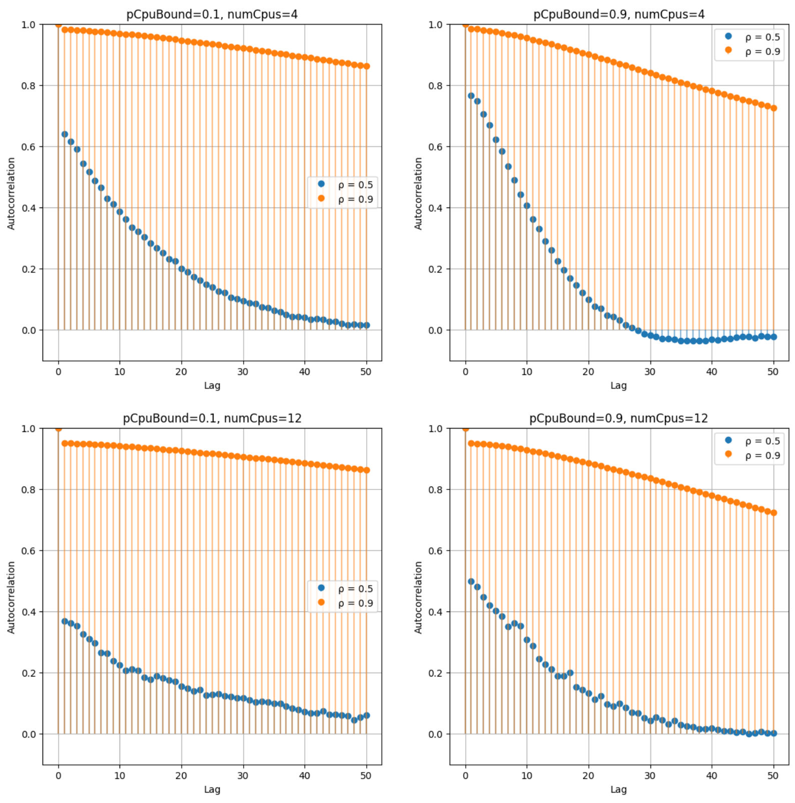


Figure 4.11: Autocorrelation plot of the waiting time of the systems.

The observation of the autocorrelation of the waiting time has been done with the lag plots in fig. 4.11. With high utilization, the correlation is always high and this reflects the expectations: if a process has waited a long time, the next one will probably wait a long time too. When the system is less utilized, the correlation is lower, since queuing is less likely.

		numCpus = 4				numCpus = 12			
		pCpuBound = 0.1		pCpuBound = 0.9		pCpuBound = 0.1		pCpuBound = 0.9	
Index		$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$
Mean	Value	2.7	92.4	5.8	213.0	0.1	21.2	0.3	43.0
	95% CI Low	2.3	81.6	4.7	186.1	0.1	18.0	0.2	36.9
	95% CI High	3.3	103.7	8.1	242.7	0.2	25.0	0.6	50.4
Std Dev	Value	7.8	108.9	21.5	252.7	0.9	35.4	2.2	64.8
	95% CI Low	6.4	99.4	14.4	221.6	0.6	30.8	1.3	56.6
	95% CI High	9.4	120.4	36.2	300.0	1.3	43.0	4.2	75.3

Table 4.3: Bootstrap results for waiting time mean and Std Dev. (ms)

The quantitative analysis of the waiting time in the exponential case is reported in table 4.3. Of course, with same duration of the processes and same utilization, the more each process spends in the CPU, the more the processes meanly waits in the queue. So with high `pCpuBound` and low `numCpus` the mean value of the waiting time increases.

Since the confidence intervals of the mean value and of the standard deviation do not overlap, we can infer that the distribution is not exponential.

CPU utilization

	numCpus = 4				numCpus = 12			
	pCpuBound = 0.1		pCpuBound = 0.9		pCpuBound = 0.1		pCpuBound = 0.9	
Index	$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$
Mean	2.02	3.61	2.03	3.63	6.01	10.82	6.03	10.80
Std Dev	1.30	0.88	1.30	0.86	2.43	1.99	2.45	2.00

Table 4.4: Mean and Std Dev of number of busy cpus.

We see that the obtained values are inline with the theoretical model:

$$\rho * numCpus = meanCpuUtilized \tag{4.2}$$

For example, in the second column, $0.9 * 4 \approx 3.61$

Ready queue length

In table 4.5 it is reported the mean and the standard deviation of the number of ready processes in the queue.

	numCpus = 4				numCpus = 12			
	pCpuBound = 0.1		pCpuBound = 0.9		pCpuBound = 0.1		pCpuBound = 0.9	
Index	$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$
Mean	0.25	13.72	0.22	10.27	0.03	9.81	0.03	7.19
Std Dev	0.96	18.47	0.89	12.99	0.29	16.34	0.28	10.57

Table 4.5: Mean and Std Dev of number of ready processes in queue.

The values are in line with the theoretical model. In fact, applying Little's Law and computing the ratio between $E[N_q]$ and the arrival rate, the values obtained are coherent with the mean of the waiting time described in table 4.3.

4.4.2 Shortest Job First

Turnaround time

From fig. 4.13 we observe that all distributions, even when $\rho = 0.9$ resemble an exponential distribution, in contrast to the `FCFS` case.

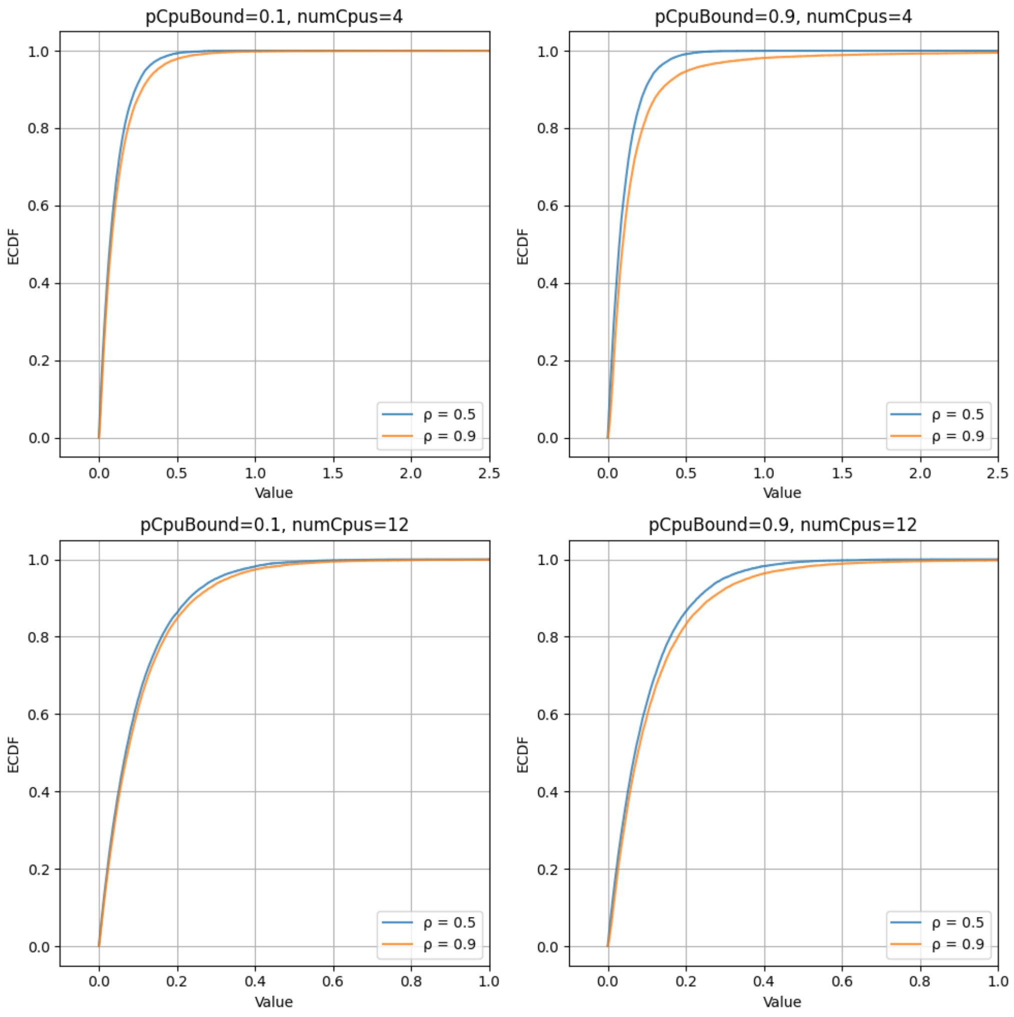


Figure 4.12: Empirical CDF of the turnaround time of the systems.

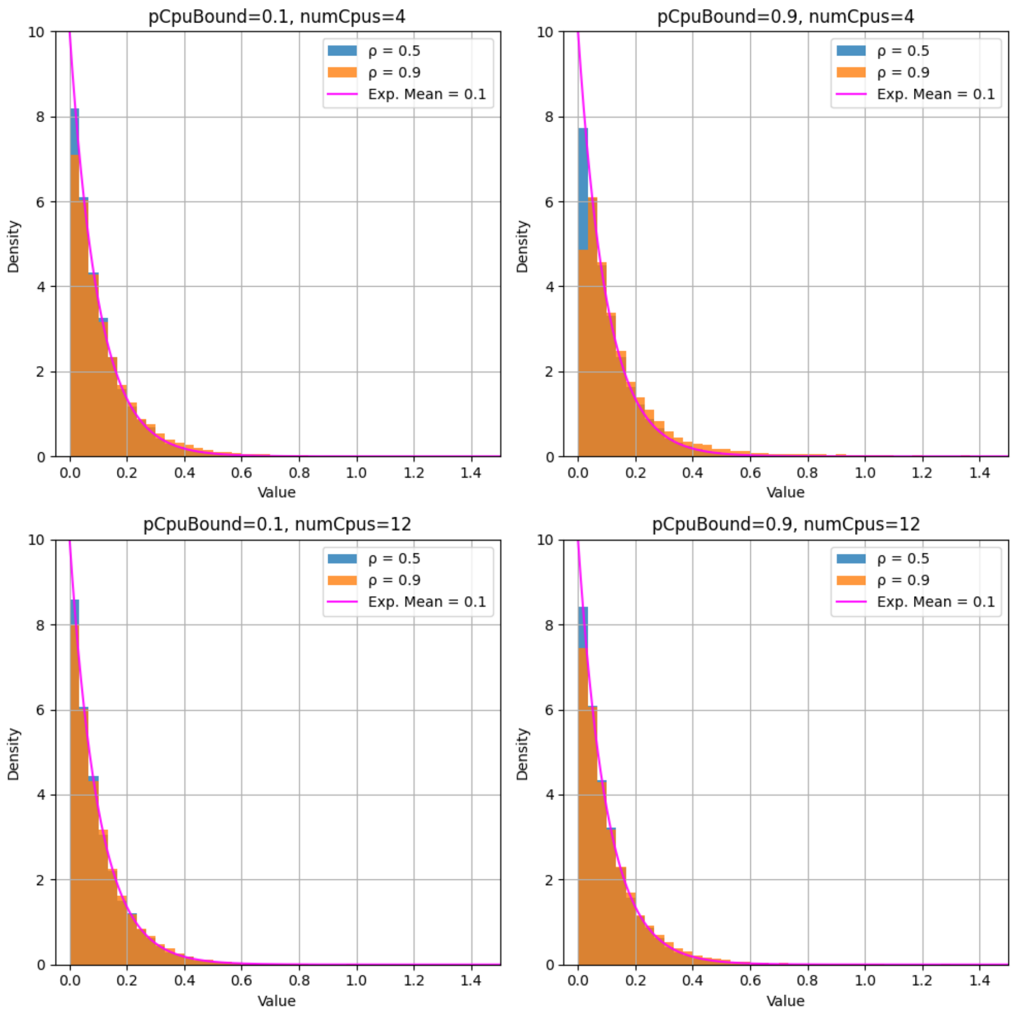


Figure 4.13: Density plot of the turnaround time of the systems.

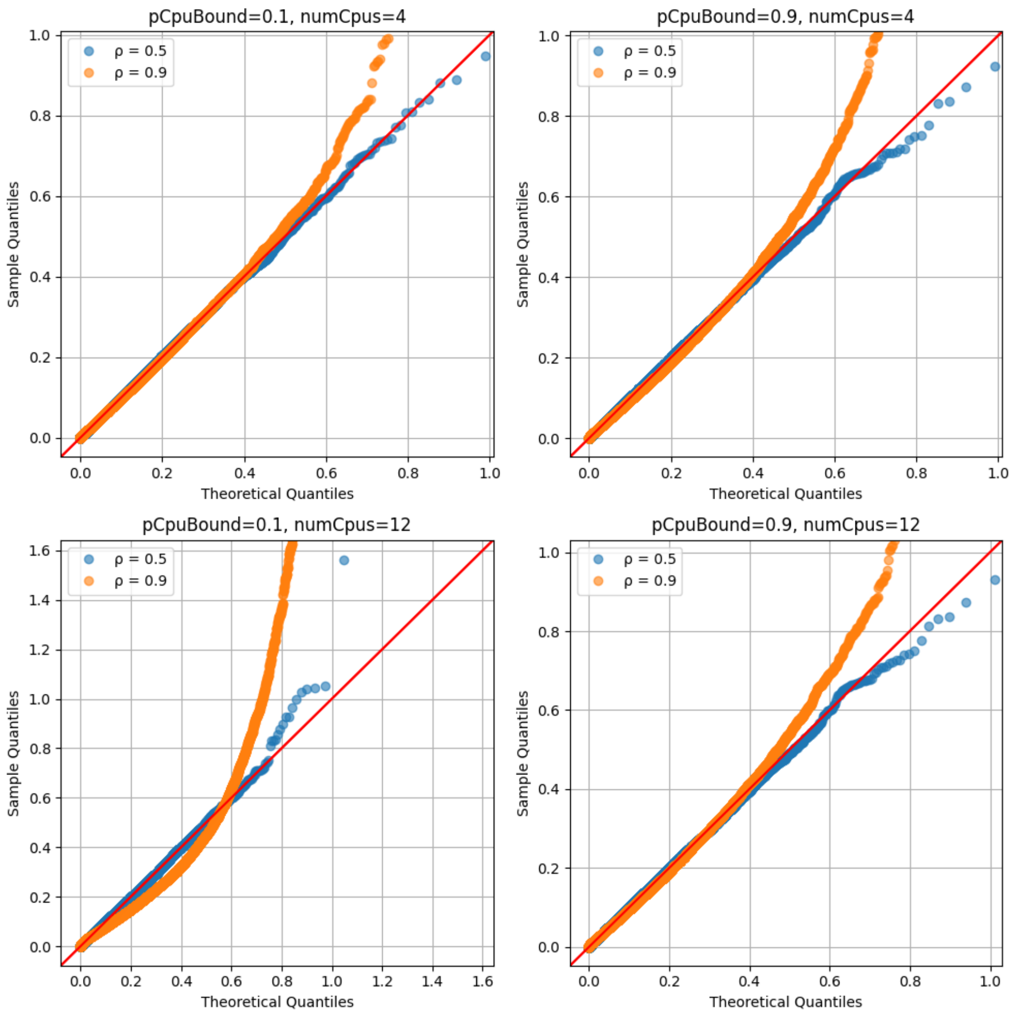


Figure 4.14: QQ plot of the turnaround time of the systems against an exponential distribution with the same mean.

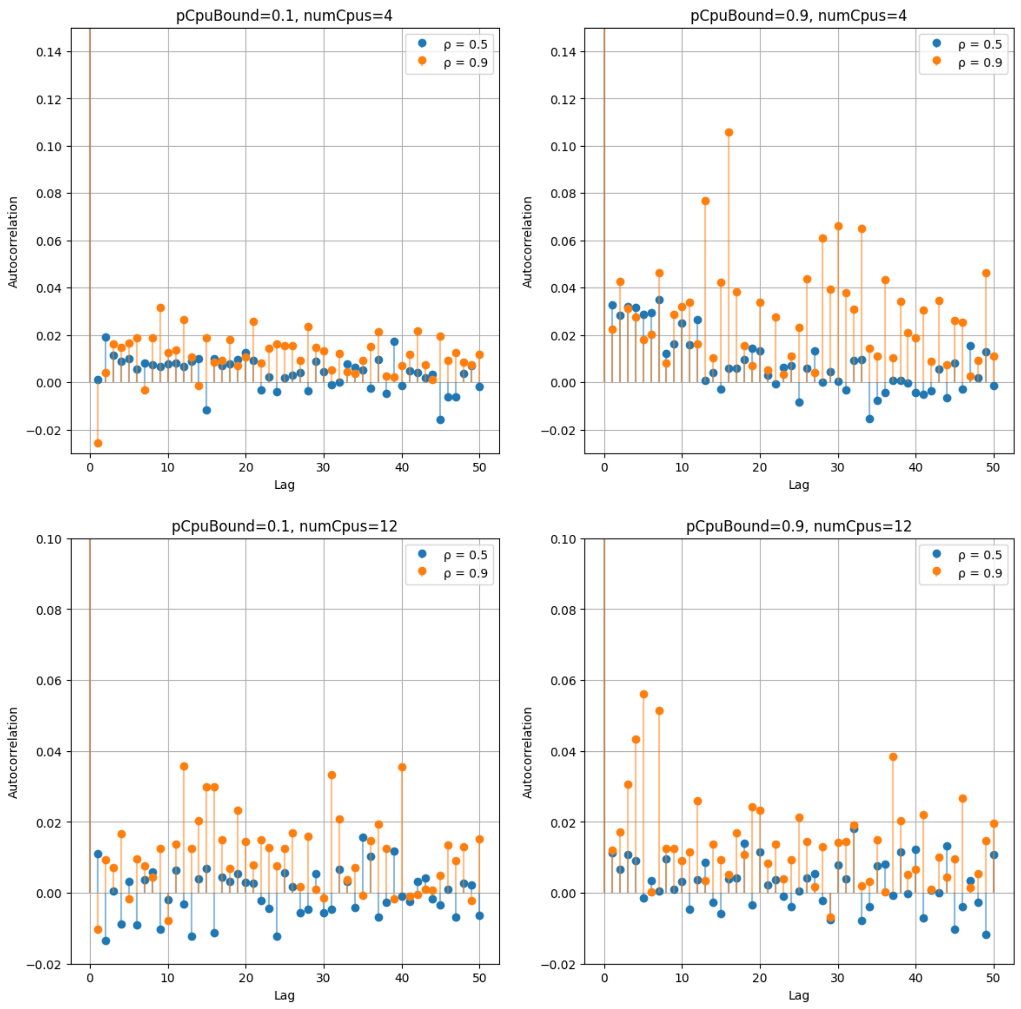


Figure 4.15: Autocorrelation plot of the turnaround time of the systems.

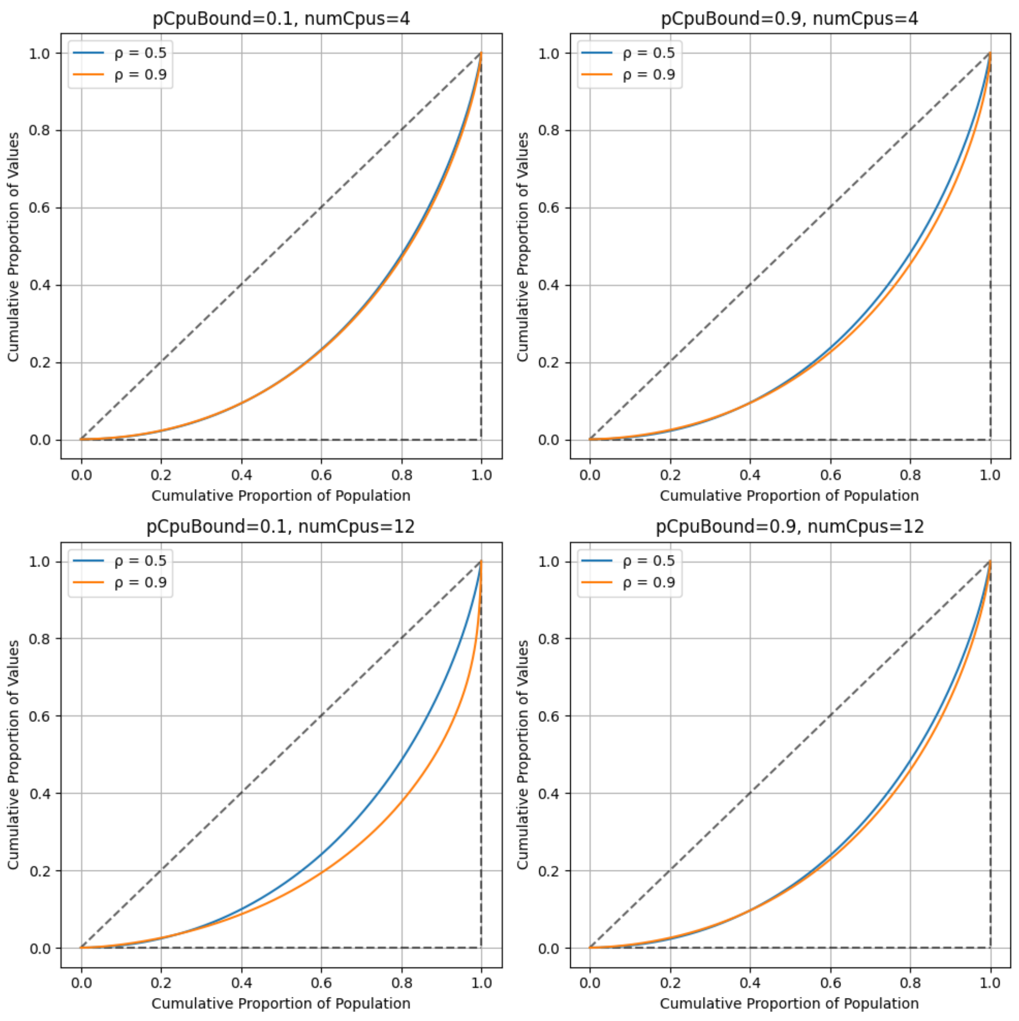


Figure 4.16: Lorenz curve of the turnaround time of the systems.

		numCpus = 4				numCpus = 12			
		pCpuBound = 0.1		pCpuBound = 0.9		pCpuBound = 0.1		pCpuBound = 0.9	
Index		$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$
Mean	Value	102.2	123.3	105.7	168.4	98.7	107.7	100.0	120.1
	95% CI Low	100.4	117.7	102.0	157.6	97.0	103.3	98.5	114.3
	95% CI High	104.3	129.6	110.0	180.7	100.7	112.8	101.2	127.6
Std Dev	Value	100.0	142.3	106.9	294.2	97.2	123.4	98.9	169.4
	95% CI Low	97.7	129.9	101.0	253.6	94.8	113.3	97.2	128.5
	95% CI High	102.8	170.4	113.8	354.2	99.8	146.7	100.8	258.6

Table 4.6: Bootstrap results for turnaround time mean and Std Dev. (ms)

Waiting time

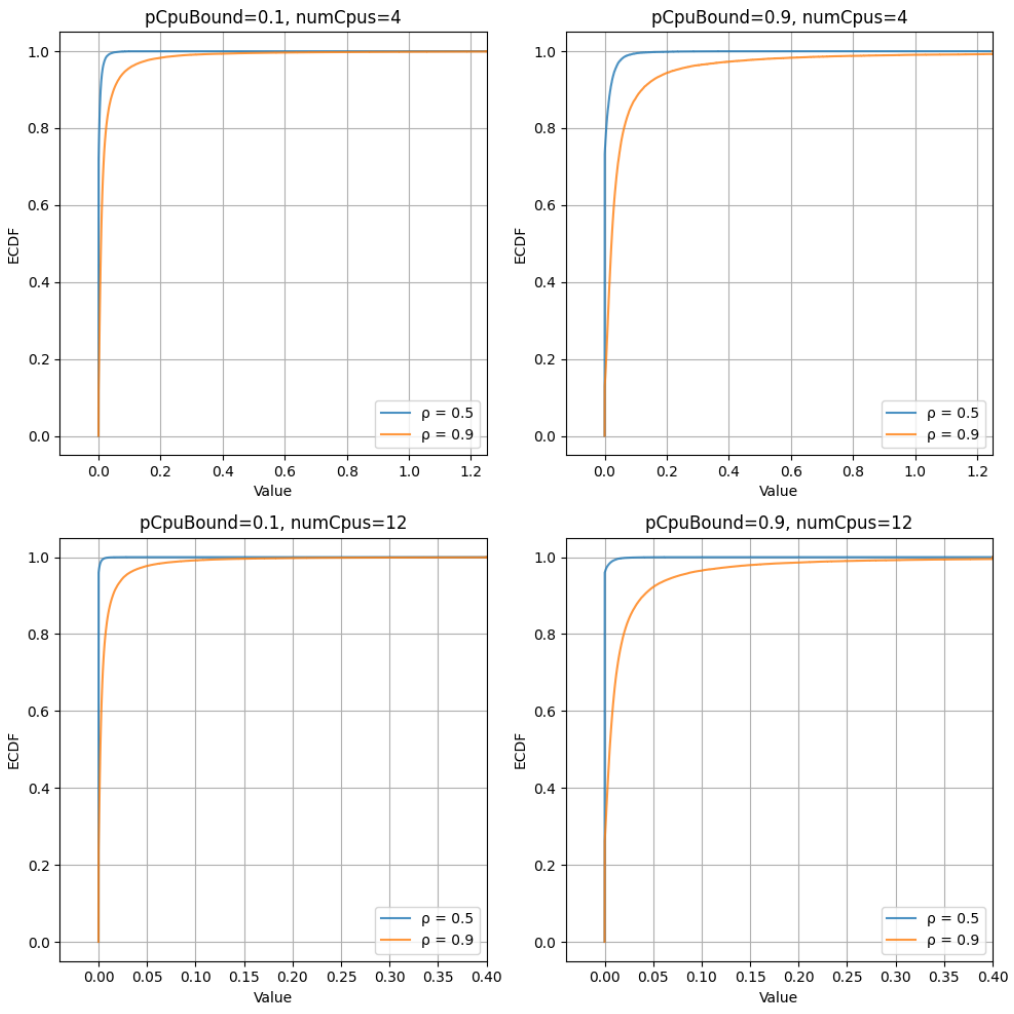


Figure 4.17: Empirical CDF of the waiting time of the systems.

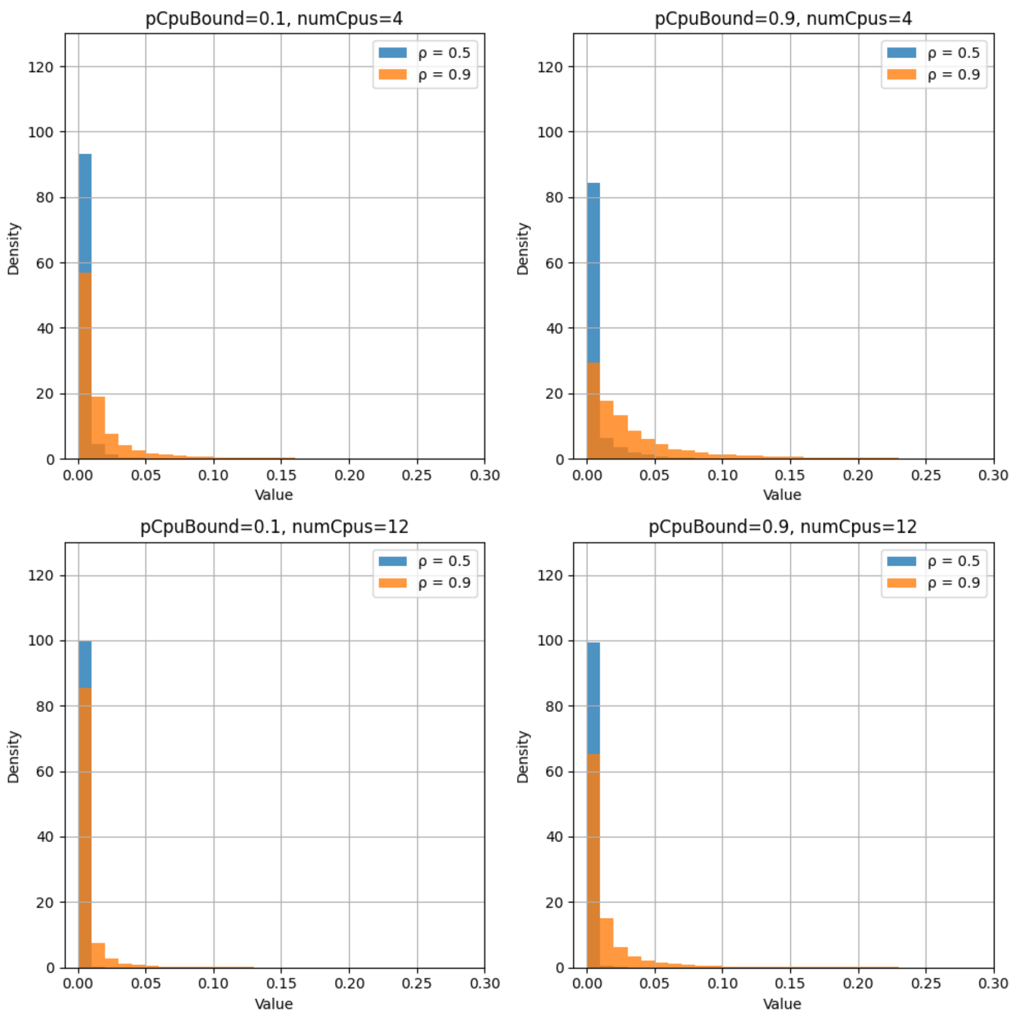


Figure 4.18: Density plot of the waiting time of the systems.

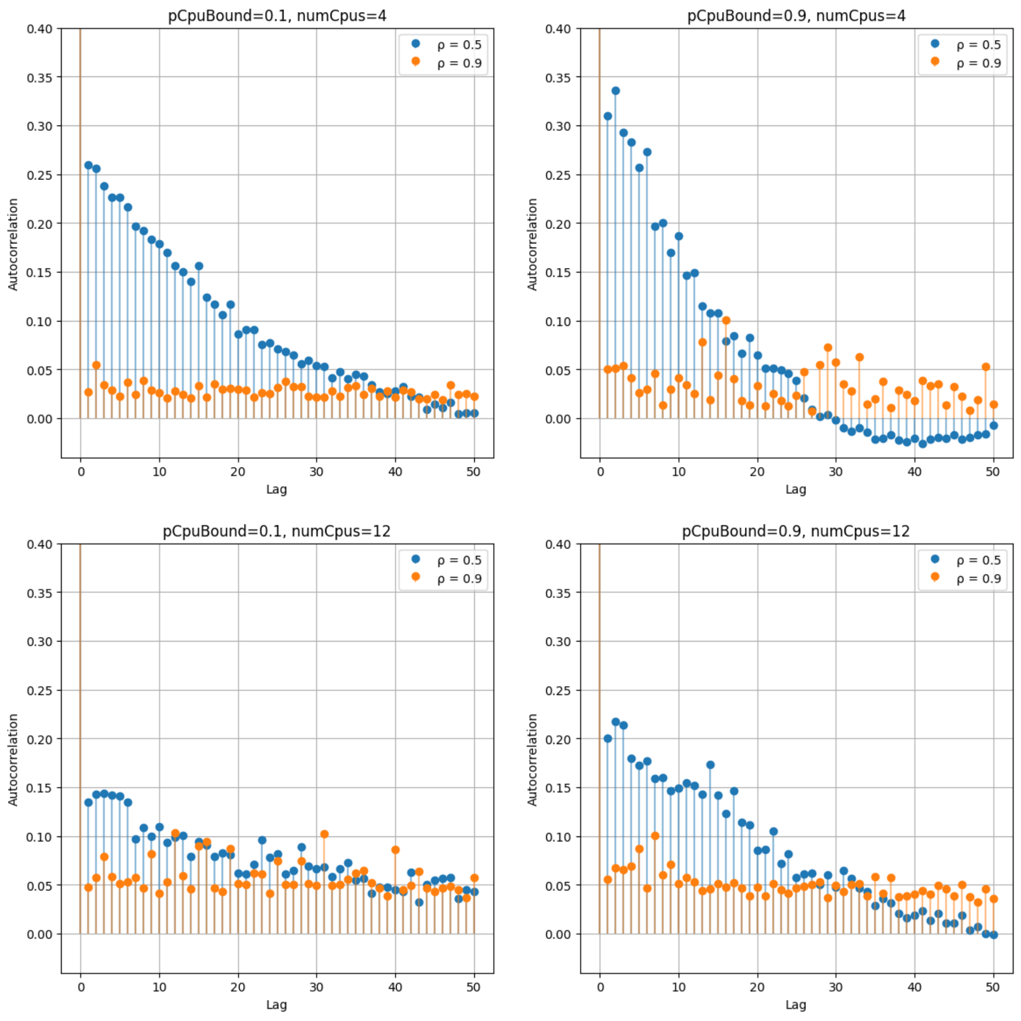


Figure 4.19: Autocorrelation plot of the waiting time of the systems.

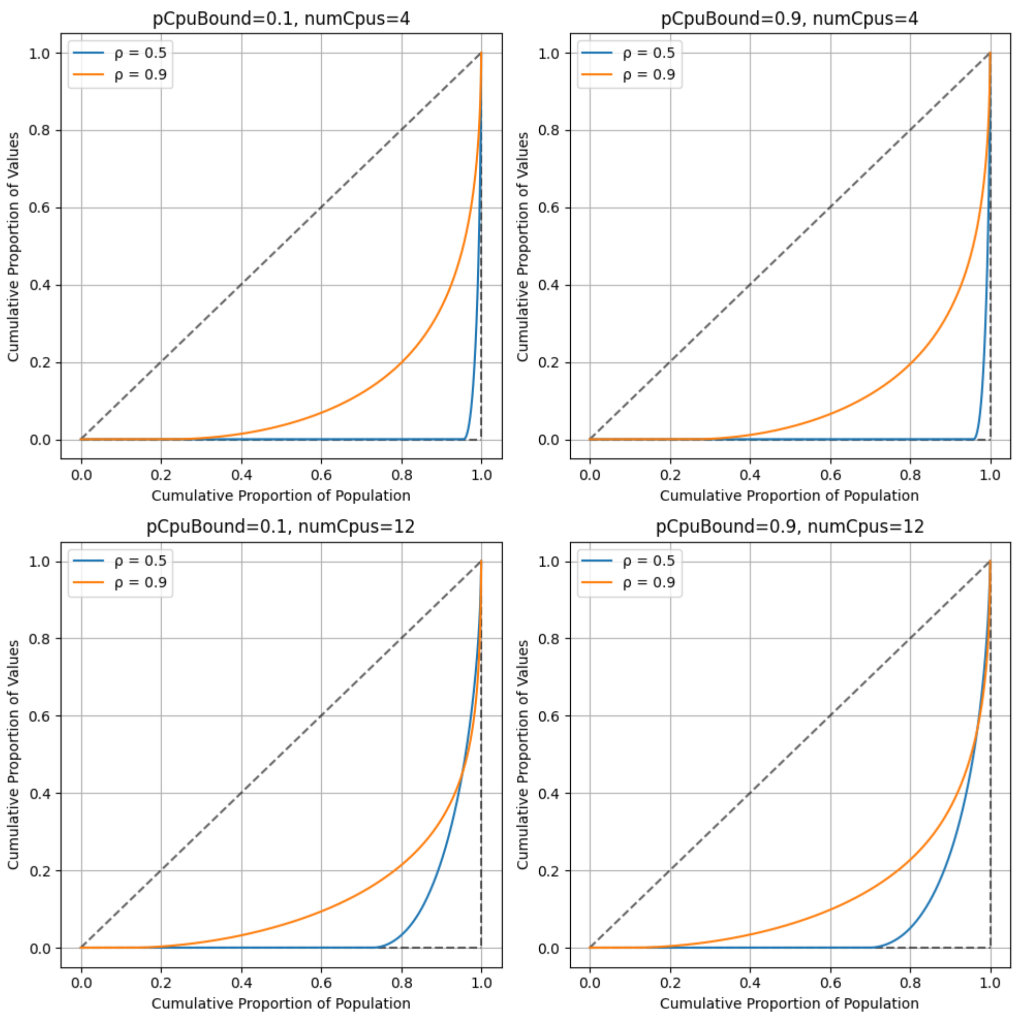


Figure 4.20: Lorenz curve of the waiting time of the systems.

		numCpus = 4				numCpus = 12			
		pCpuBound = 0.1		pCpuBound = 0.9		pCpuBound = 0.1		pCpuBound = 0.9	
Index		$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$
Mean	Value	2.2	26.4	4.5	78.5	0.1	6.2	0.2	18.9
	95% CI Low	1.9	23.6	3.6	67.4	0.1	5.4	0.1	16.9
	95% CI High	2.8	32.1	5.7	105.5	0.1	7.4	0.3	23.9
Std Dev	Value	6.7	112.6	13.2	430.3	0.5	19.3	1.7	86.7
	95% CI Low	5.1	72.4	10.9	260.8	0.4	14.2	1.1	46.5
	95% CI High	10.3	192.6	18.8	801.1	0.6	28.0	3.2	177.7

Table 4.7: Bootstrap results for waiting time mean and Std Dev. (ms)

CPU utilization

There is no difference with section 4.4.1 since the `FCFS`and `SJF` schedulers have the same behaviour in terms of CPU utilization.

Ready queue length

	numCpus = 4				numCpus = 12			
	pCpuBound = 0.1		pCpuBound = 0.9		pCpuBound = 0.1		pCpuBound = 0.9	
Index	$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.9$
Mean	0.17	3.54	0.16	3.47	0.02	3.12	0.02	2.93
Std Dev	0.66	3.62	0.62	3.57	0.23	4.03	0.23	3.80

Table 4.8: Mean and Std Dev of number of ready processes in queue.

5 Conclusions

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

```cpp
// Your First C++ Program

#include <iostream>

```

```cpp
int main() {
    std::cout << "Hello World!";
    return 0;
}
```

Listing 5.1: `test.cpp`

Nunc sed pede. Praesent vitae lectus. Praesent neque justo, vehicula eget, interdum id, facilisis et, nibh. Phasellus at purus et libero lacinia dictum. Fusce aliquet. Nulla eu ante placerat leo semper dictum. Mauris metus. Curabitur lobortis. Curabitur sollicitudin hendrerit nunc. Donec ultrices lacus id ipsum.

[**Persona 1: Commento 1**]

[**Persona 2: Commento 2**]

Test cite[**GOOGLE**].

Test footnote[1].

Figure 5.1: Un gattino.

Ciao

Nunc sed pede. Praesent vitae lectus. Praesent neque justo, vehicula eget, interdum id, facilisis et, nibh. Phasellus at purus et libero lacinia dictum. Fusce aliquet. Nulla eu ante placerat leo semper dictum. Mauris metus. Curabitur lobortis. Curabitur sollicitudin hendrerit nunc. Donec ultrices lacus id ipsum.

[1] Footnote