Process Scheduling Algorithms' Workload Design, Simulation and Analysis with Experimental Data Analysis Techniques

Experimental Methods in Computers Science
Department of Informatics Engineering
College of Science and Technology
Universidade de Coimbra

Oleksandr Yakovlyev 2015231448
Achilles S. Do Nascimento 2017206098
Pedro F. Durão B. L. Félix 2017276005

Abstract

In this assignment we compare the performance of four process scheduling algorithms, First Come First Served (FCFS), Round Robin (RR), Shortest Job First (SJF) and Shortest Remaining Time First (SRTF) under different workloads. We generate different workloads and compare the performance of the different scheduling algorithms using mainly visual techniques of experimental data analysis.

Keywords: Scheduling Algorithms, Statistics, FCFS, RR, SJF, SRTF, Experimental Data Analysis.

Method

In this section we will give an overview of how we structured our work. We began our assignment with research on cpu scheduling algorithms. Our primary focus was to discover what kind of workloads existed and what kind of cpu scheduling was more efficient for each type of workload. We also experimented with the recommended linux perf tool to gain some insight of the usual cpu loads, but ultimately all of our workloads were generated with the `gen_workload.py` script. At this stage we already had a few hypotheses in mind, so we ran the simulations and applied experimental analysis methods to the generated data.

In the next section we will describe our method, including the code and tools used. In the section after, we will enumerate our hypothesis and present the resulting data and graphs that we generated to confirm our assumptions while discussing our results. Finally we will state our conclusions about this type of analysis.

Generating workloads and running simulations

To streamline this process we created a bash script `run_batch_simulations.sh`. As input it takes a name for the workload, followed by all the standard input parameters for the `gen_workload.py` script.

While taking initial tests we opted for a single parameter variation. This enabled us to gain some insight on how the input parameters influence the simulations. We also started testing workloads with 5 different seeds per test, but because of the high similarities in the results between the seeds, we shortly reverted to single seed runs.

Workload and Scheduler Analysis

To study the different workloads and schedulers we utilized the recommended R language. We began by loading the data into dataframes, from which we generated several graphs: barplots, density estimates, qq plots, boxplots and heatmaps of 2-d bincounts. To generate the graphs we used the ggplot package. When comparing workloads we looked for variables that were common between all the schedulers, such as total cpu time and total io time, often visualizing them as shown in figures 1 and 4. Now when comparing schedulers we looked at the turn around times, ready wait times and io wait times.

Hypothesis Taken

After gaining some understanding of the differences of the possible workloads and investigating the characteristics of the four schedulers in question we accumulated some hypotheses that we tried to prove. These are listed below, in no specific order:

1. Under high CPU load, the RR scheduler will have a higher median turnaround time
2. FCFS will have a lower overall turnaround time and ready wait time than RR
3. SRTF will behave like a SJF when all the processes are already in the ready queue
    3.1. SRTF will have a better median turnaround time than SJF if the first processes are heavy on the cpu.
4. If the arrival of processes is sufficiently sparse all the schedulers will behave closely to FCFS
5. IO wait time will be equal to IO burst time in a environment where the CPU is the bottleneck
6. If the processes in the ready queue follow a normal distribution in terms of CPU time, the distribution of the waiting time in the ready queue of the RR scheduler will also be normal

Data collecting and result analysis

In this section we will guide through the work we did through the assignment, explaining our procedures, displaying relevant graphs and tables and our conclusions about the results.

Our firsts series of tests were modeled around a cpu bottlenecked scenario. We began by defining default values for the workloads, then we varied only the min and max cpu parameters (and also the quantum parameter for the b3 and c tests to reduce the simulation time).

Default values	
num_procs	1000
mean_io_bursts	10
mean_iat	10
min_io	1
max_io	2

cpu_test	min_cpu	max_cpu	quant
a1	0.1	0.2	0.01
a2	0.1	1	0.01
b1	1	2	0.01
b2	1	10	0.01
b3	1	100	0.1
c	10	20	0.1

Table 2. Default values for the cpu_tests family of workloads (left) and specific differences (right)

We then created a hexagonal heatmap of 2d bin counts to visualise the distribution of processes by total cpu time and IO time (Figure 1). As we can see, the IO time was the same for all workloads. We also made sure the workloads were generated with the same seed, so the number of cpu bursts per each process is also the same. The only variation was the duration of each cpu burst.

Then we analysed the data with the summary and aggregate R functions, but visual techniques such as density plots and boxplots presented a clearer representation as seen in Figure 2 and 3.

In figure 2 we show the distributions of the turnaround time using density plots. As we can see, the cpu_tests_a1 and cpu_tests_a2 are identical. This is because the cpu burst times are so sparse in between processes that all the schedulers operate on a first come first served basis. This is also visible in Figure 3, a box-plot of the time that the processes spent on the ready queue.

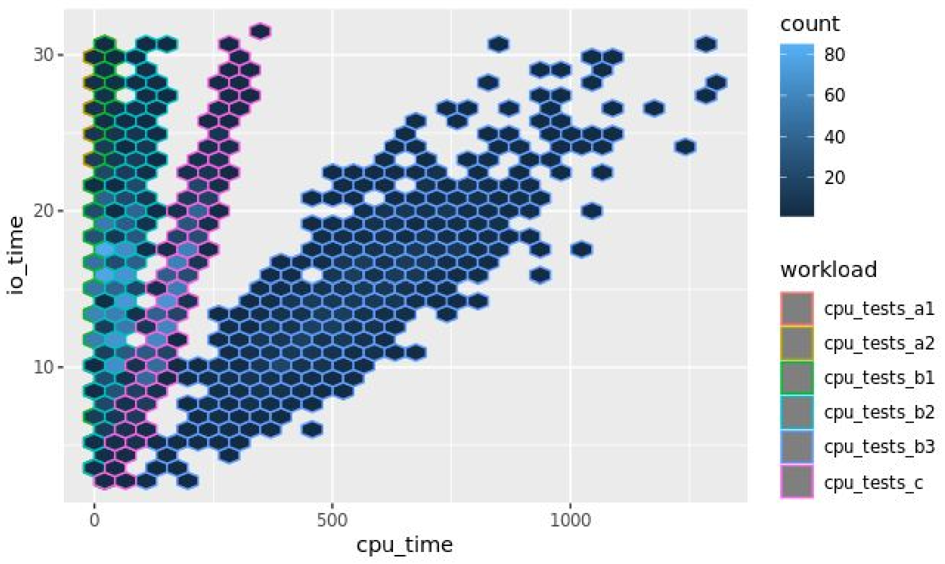


Figure 1. Hexagonal bincount comparing workloads by total cpu_time/io_time per process

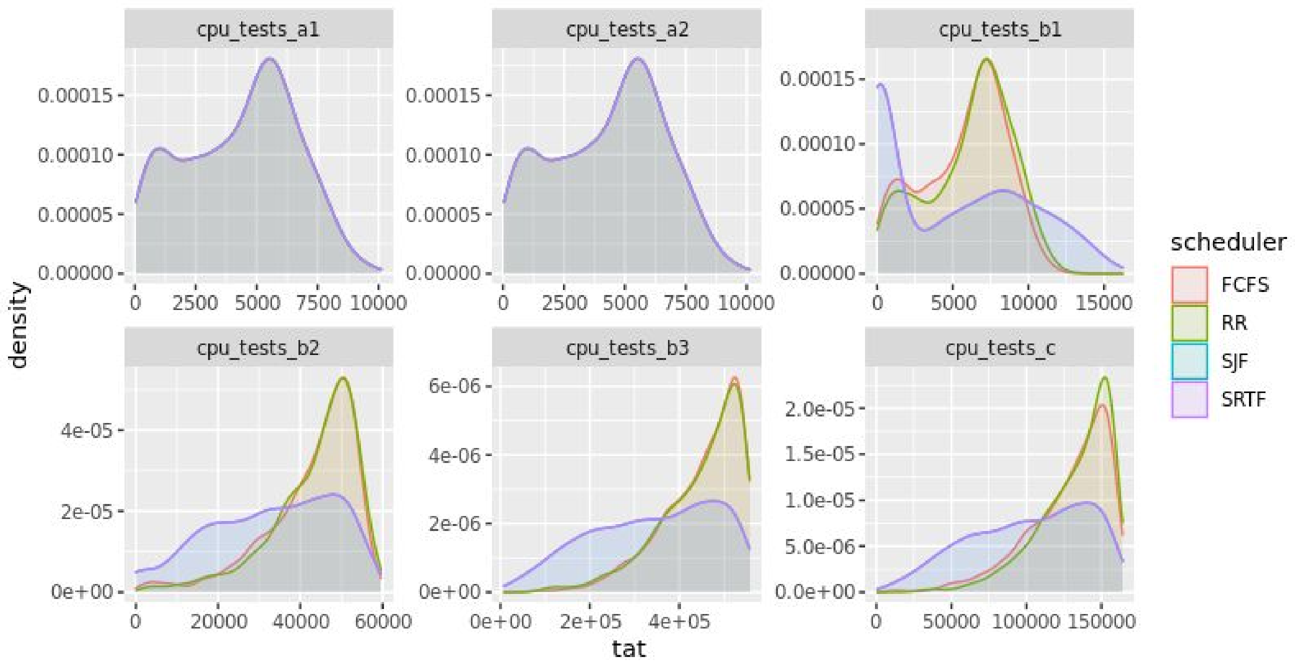


Figure 2. Density plots comparing the turnaround type between the cpu_tests family of workloads

Both Figure 2 and 3 show us that the RR and FCFS schedulers tend to have a narrower distribution of turn around and ready wait times, with long left tails. To confirm this with more precision we did a kurtosis and skewness test on the cpu_tests_b3 and got the following results:

scheduler	kurtosis	skewness
FCFS	0.8955963	-1.0833338
RR	1.0492609	-1.1439216
JFS	-0.9571870	-0.3254707
SRTF	-0.9573830	-0.3253192

Table 2. Kurtosis and skewness for each scheduler in workload cpu_tests_b3

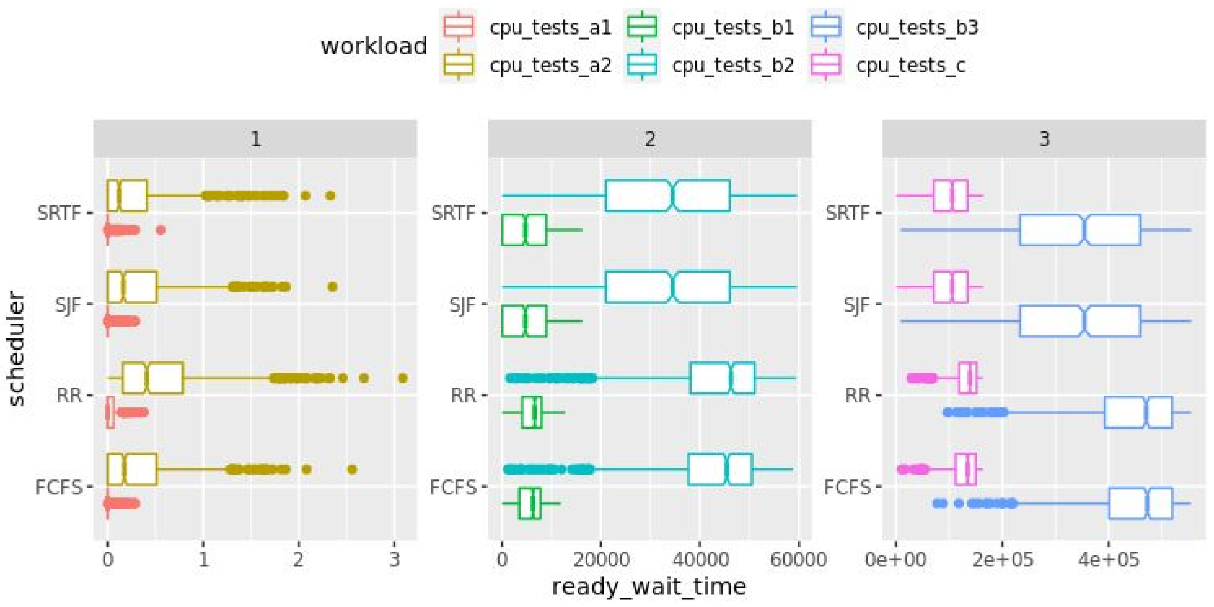


Figure 3. Boxplot for the ready_wait_times for the cpu_tests family of workloads.
These are separated in three groups for scaling reasons.

We also noticed that SJF and SRTF have nearly identical behaviour, and have a much wider range of values. This makes sense from a practical point of view: SJF and SRTF behave equally because, in all the scenarios above, the processes quickly pile up in the ready queue and the SRTF cannot leverage it's preemptive behaviour. All the jobs are processed in a SJF manner, minimizing the average turnaround time and maximizing throughput.

After these results we had the following question: why do FCFS and RR have similar results? This went against the second hypothesis. Even though RR *seemed* to have the TAT concentrated to the right end of the spectrum, it was by a small margin. We believe it was because the minimum value for the CPU bursts was relatively low, so there were always small processes that RR was able to finish. To confirm this we generated another test case, "big and fast", with low burst counts (1) , high cpu times (10-20) and low arrival times (1).

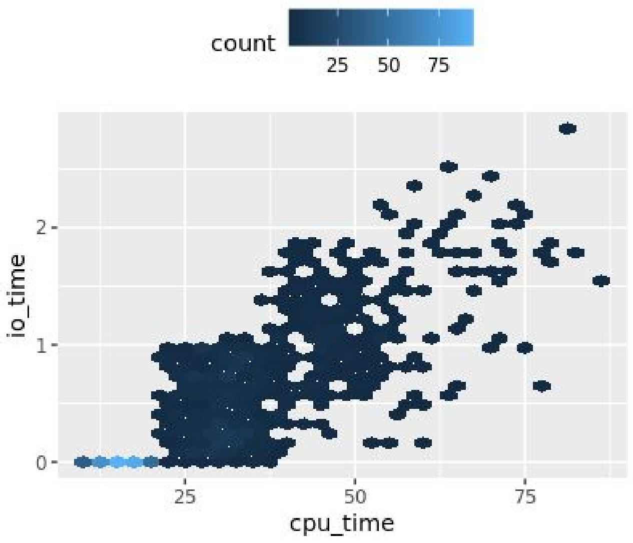


Figure 4. Hexagonal bincount showing the process distribution by total cpu_time/io_time per process

This once again created a cpu bottleneck scenario. The ready wait time and turnaround time (TAT) were near identical, and shown in the following graphs:

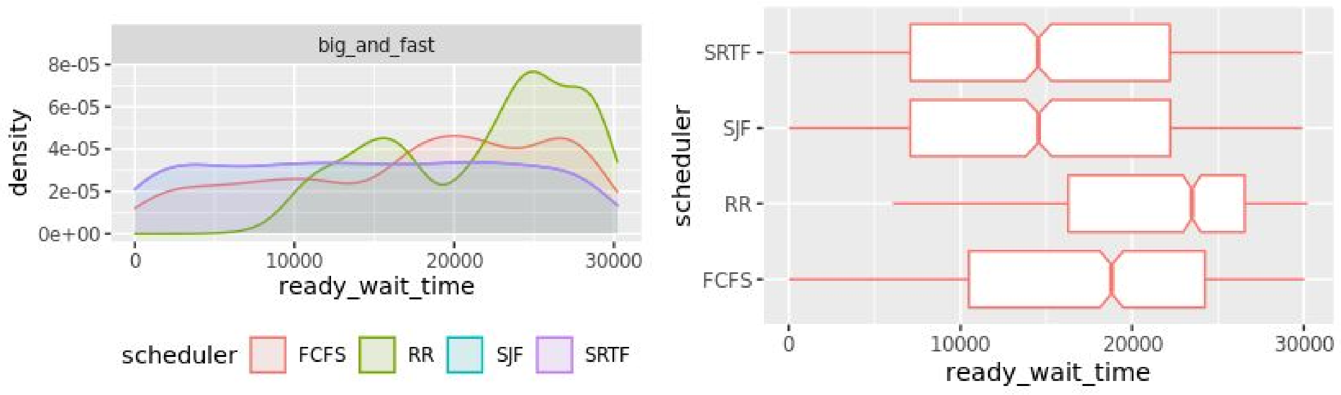
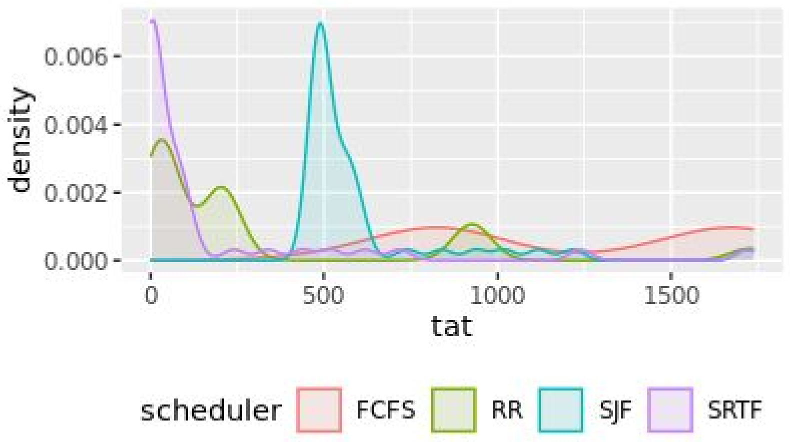


Figure 5. Density (left) and Boxplot (right) plots for the ready_wait_time of the big_and_fast workload

From the density plot, we see that RR did in fact hold the processes in the ready queue more time than the other schedulers (also visible on the boxplot), confirming our second hypothesis.

During these experiments, we noticed that the SJF and SRTF were having the same TAT and RWT times, so we designed a workload that would evidentiate the differences between the two schedulers.
The following workload, named “inverse”, is a synthetic workload, meaning that there was no variation and no randomness. It was handcrafted specifically for this test. It is as follows: the processes arrive in exactly one second intervals, with only one CPU burst[1]. The first has one cpu burst of 500s, followed by 3 processes with one 100s cpu burst, 6 with 10s and finally 10 with 1s. This sequence is repeated two times. By our intuition, the SJF would be stuck working on the first long process because of its non-preemptive nature, while the SRTF would immediately switch for the more recent but shorter process.



[1] Actually we have two CPU bursts and one IO burst, but the first two have a negligible duration of 0.000001s. We did this workaround because we noticed that the turnaround time was not being counted correctly from the time of arrival of the process. We have some tests that illustrate this and will ask the teacher about this in the next practical class.

Figure 6. Density plot comparing turnaround time between schedulers for the synthetic workload

Figure 6 displays exactly that. SJF turnaround times peaked by the 500s mark, finishing the first 500s process and quickly switching to the 1s, then the 10s then the 100 seconds (the bumps in the 600-1250 range) and finally the last 500s process (at ~1750). SRTF on the other hand behaved as intended and always prioritized the shortest process.

RR also shows an interesting distribution: there are 4 peaks: one for each process group (1s, 10s, 100s and 500s).

To test our sixth hypothesis we made another workload, the “normal”. This test has the following parameters: 10000 processes, 10 mean io bursts, exactly 10s of CPU and IO durations.

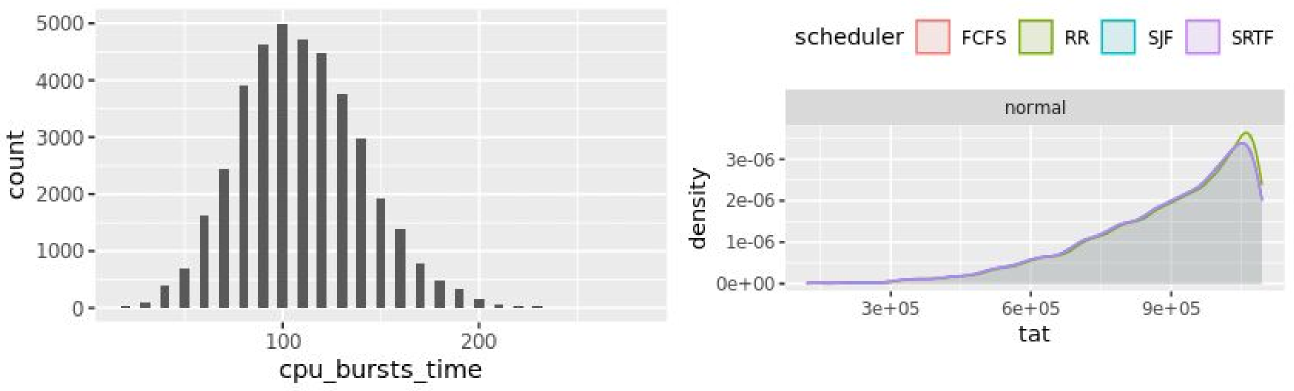


Figure 7. Total cpu bursts time histogram for the “normal” workload (left) and turnaround time density plot (right)

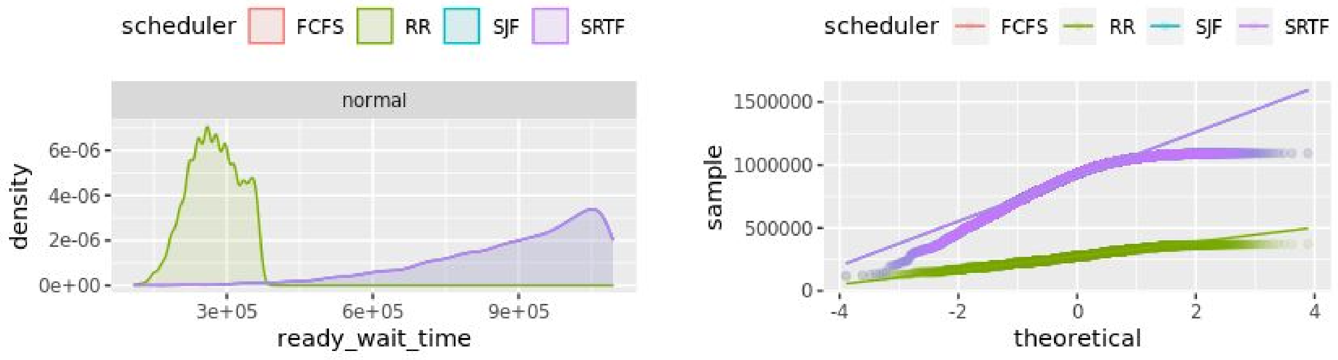


Figure 8. Density plot for the ready wait time (left) and a Q-Q plot for the same data (right)

As expected, all but the RR scheduler had equal TAT, because all the cpu bursts had the same duration, but even so the differences are muffled at this scale (10000 processes). The interesting difference is hidden in the ready wait time: because the RR is constantly switching contexts (1s quantum) the ready wait time is considerably lower than of the other schedulers, appearing to also follow a normal distribution.

Conclusions

Taking everything into account, all the hypotheses that were put into consideration seem to be valid, at least judging by the different visual tests and analysis methods made throughout this assignment. As mentioned before, different seeds didn't seem to affect results in a meaningful way (Figure 9).
With this we were also able to understand why this analysis method is used in order to take quick results. Despite not having the same degree of formality as the classical statistical approach, the simple visualization and quick analysis of the results allows to formulate hypotheses in an iterative way, enabling a quick return to the start when faced with a wrong progression path.

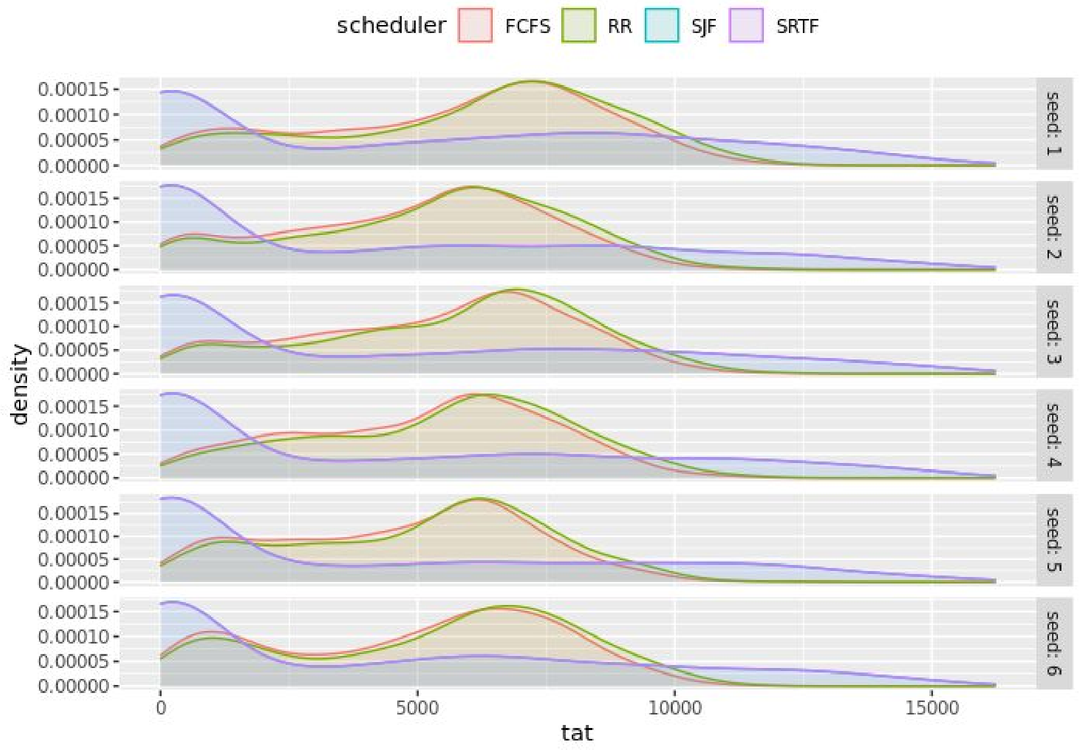


Figure 9. Differences in the turnaround time of the cpu_tests_b2 for different seeds.

References

Simpson, G. (2006). Introduction to R and Exploratory data analysis.
Anushka, V. (2020). Difference between SJF and SRJF CPU scheduling algorithms. https://www.geeksforgeeks.org/difference-between-sjf-and-srjf-cpu-scheduling-algorithms/
Tutorialspoint, (2015) Operating System Simply Easy Learning, https://www.tutorialspoint.com/operating_system/index.html
Tinyverse. https://ggplot2.tidyverse.org/reference/index.html