Simulating Scheduling Algorithms Using Event-Driven Simulation

Delgersaikhan Bayarsaikhan
Whitworth University
Spokane, Washington, 99251
dbayarsaikhan23@my.whitworth.edu

Michael Lie
Whitworth University
Spokane, Washington, 99251
mlie23@my.whitworth.edu

Nirjal Shakya
Whitworth University
Spokane, Washington, 99251
nshakya23@my.whitworth.edu

***Abstract*—This paper presents a comprehensive investigation of different optimization algorithms for discrete event simulations. Through a series of case studies and performance benchmarks, we evaluate the trade-offs between computational speed and simulation fidelity.The results of our study demonstrate the effectiveness of optimization algorithms in reducing simulation execution times. We observe that event-driven algorithms excel in scenarios where event handling dominates computational costs, while process-driven algorithms offer advantages in situations involving complex state transitions. Hybrid algorithms, combining the strengths of event-driven and process-driven approaches, exhibit promising performance across a broad range of simulation scenarios.**

1. Introduction

The idea of a discrete event simulation has been around since 1965 and over the next half-decade, we utilized this method in all fields such as healthcare, manufacturing, etc. In recent years, discrete event simulations (DES) have become an indispensable tool for modeling and analyzing complex systems across various domains, ranging from manufacturing and logistics to healthcare and transportation. These simulations enable researchers and practitioners to study the behavior and performance of systems under different conditions, providing valuable insights for decision-making and performance optimization. However, as the complexity of systems increases, so does the computational burden of running these simulations. Therefore, the development of efficient algorithms and techniques for optimizing discrete event simulations has become a pressing research challenge. The optimization of discrete event simulations involves finding strategies to minimize the time required to execute simulations while maintaining accuracy and reliability. Traditional simulation approaches may suffer from long execution times, especially when dealing with large-scale and intricate systems. Consequently, researchers have focused on developing novel algorithms and methodologies that can expedite simulation processes, thereby enabling faster analysis and exploration of system behaviors. By applying these algorithms to a series of case studies and performance benchmarks, we aim to assess their efficiency, accuracy, and applicability to real-world problems. Furthermore, we will analyze the trade-offs between computational speed and the fidelity of simulation results. Our findings will provide valuable insights into the strengths and limitations of different optimization algorithms, guiding researchers and practitioners in selecting the most suitable approach for their specific simulation requirements. For this simulation to operate in real-world operations, we have to consider all sorts of situations and possibilities so that all processes can execute at efficient times. Therefore, our purpose is to model a discrete event simulation using real-life data and simulate it on different algorithms to see which one is the most optimal. Furthermore, to make the simulation mimics a real-life computer, different factors such as I/O time and Switch Time were referenced based on average modern computers performance. To find the best algorithm, we have created 14 different algorithm combinations including multi-level queues and single-level queues. From those scenarios, different graphs were created to compare different factors of an algorithm's efficiency, including Average Throughput (ATht), Average Wait Time (AWT), Completed Jobs in a certain time period (CJ), and Average Processor Utilization (CPU Util).

2. Preliminaries

Event-driven simulation deals with scheduling events and simulating their occurrence in discrete time steps. In this paper, we are focusing on event-driven simulation with scheduling algorithms to compare the performance of different scheduling algorithms. A few terminologies that'd be beneficial to understand before proceeding are mentioned below:

- **Discrete Event Simulation**: A simulation where the system is modeled as a series of discrete events, each of which has a time stamp associated with it.
- **Future Event List**: A list of events that are scheduled to happen in the future, sorted based on the time stamps.
- **Event Driven Simulation**: A simulation technique where the system state changes only when an event occurs.

- **Scheduling algorithms**: Algorithms used to schedule different processes to be assigned to the CPU.

3. Proposed Schemes, Methods, and Algorithms

It is evident from previous research that future event list-based simulation techniques result in significant performance improvements over other simulations. Moreover, these simulations have been used to compare the performance of different scheduling algorithms. However, oftentimes, the simulations are time-based, which could have low accuracy and be computationally more expensive than event-driven simulations with a future event list. Moreover, this paper aims to explore more different scheduling algorithms along with the event-driven simulation.

3.1. Schemes and Algorithms

Among the many scheduling algorithms, we'll be using First-Come First-Served (FCFS), Round Robin (RR), and Shortest Process Next (SPN). [2]

- **First-Come, First-Served**: It functions in a first-in, first-out (FIFO) scheme, where each process becomes ready, joins the ready queue, and executes fully. It performs better for long processes than for shorter ones. [2]
- **Round Robin**: It uses the time-slicing technique, where each process is given a time slice before being preempted. A clock interrupt is generated at periodic intervals. When an interrupt occurs, the currently running processes are placed in the ready queue, and the next ready job is selected on an FCFS basis. [2]
- **Shortest Process Next**: It is a non-preemptive policy, where the process with the shortest expected processing time is selected next. Thus, the list is automatically sorted based on the shortest burst time. [2]

We'll be exploring the algorithms and their performance by comparing them individually, adjusting them to multiple levels, and random adjustments. The scheduling criteria metrics like avg. wait time (AWT), avg. throughput time (AThP), avg. turnaround time (ATP), avg. response time (ART), and CPU utilization will be used to measure the performance of these algorithms.

3.2. Methods

The simulations were conducted with 14 different scenarios based on 7 different algorithms and the same 100 jobs placed in a future events list as outlined below.

- Round Robin (time quantum: 16), Round Robin (time quantum: 32), First Come First Serve.
- Round Robin (time quantum: 16), Round Robin (time quantum: 32), Shortest Processor Next.
- First Come First Serve
- Shortest Processor Next
- Random Queue Iteration (Round robin (time quantum: 16), Round Robin (time quantum: 32), Shortest Processor Next, First Come First Serve)
- Round Robin (time quantum: 16)
- Round Robin (time quantum: 32)

Each of these algorithms' performance was tested with 2 different processor variations:

- 4 processors
- 8 processors

Additionally, we will also be testing a scenario as mentioned in the "An Efficient Randomized Algorithm for Real-Time Process Scheduling in PicOS Operating System" paper. It hypothesizes that when a FCFS algorithm is used in comparison with a random iteration algorithm that randomly chooses between FCFS and SPN, the average throughput and wait times for jobs using FCFS will always be higher than those using random iteration.

3.3. Assumptions

- The simulation's CPU burst time was designed through mimicking the real-life CPU burst time ranging from 10-100 milliseconds.
- The process switching latency is placed to 2 ms, based on Chen's paper. [3]
- The Average I/O time is 5ms based on Larsen's paper to mimic the real-life CPU. [4]
- To prevent job starvation, each job has an age variable associated with it, which will be incremented when a ready job is not executed in any queues. Every time a task is executed, the moveQueues() function will always be called to monitor if an event has stayed more than a certain amount of time in a queue. If it passes the threshold, any job that currently resides in queues other than the Round Robin 1 queue will be pushed back to the Round Robin 1 ready queue. By implementing this method, starvation can be prevented. This age variable is implemented in all multi-level queued algorithms except the Random Iteration Algorithm.
- Different numbers of processors were used to test the efficiency and how the algorithms will perform (4 processors, 8 processors).
- There are cases where Arrival or Departure time may overlap. To prevent this problem, priority queue systems are introduced to the simulation.
- During I/O Operation, processes are moved to the waiting queue and moved back to the ready queue upon I/O completion.

4. Experimental Results

Based on the simulation, the tables and graphs below analyze the results based on the 14 scenarios.

4.1. Overall Results

Configuration	Processors	Jobs	Total Simulation Time	Avg. Throughput	Avg. Wait Time	Avg Response Time	Avg. Turnaround Time	Avg. Utilization for Processors	Completed Jobs
[4] RR16 RR32 SPN	4	100	1945	0.0514	769.21	73.52	1430.06	94.95%	59
[4] RR16 RR32 FCFS	4	100	1926	0.0519	866.78	40.33	1398.82	95.89%	64
[4] RR16	4	100	1926	0.0519	786.17	40.33	1473.91	95.89%	54
[4] FCFS	4	100	1941	0.0515	935.34	41.9	1048.26	95.14%	79
[4] SPN	4	100	1975	0.0506	67.91	680.59	788.65	93.51%	88
[4] RR32	4	100	1931	0.0518	822.64	41.58	1450.59	95.64%	55
[4] Random Iteration	4	100	1965	0.0509	853	66.15	1074.76	93.98%	87
[8] RR16 RR32 SPN	8	100	1003	0.0997	384.82	41.95	709.6	92.06%	60
[8] RR16 RR32 FCFS	8	100	972	0.1029	426.68	21.69	705.77	95.00%	63
[8] RR16	8	100	967	0.1034	392.59	21.69	738.16	95.49%	54
[8] FCFS	8	100	994	0.1006	445.5	24.88	540.89	92.89%	76
[8] SPN	8	100	1015	0.0985	65.38	314.76	415.46	90.97%	85
[8] RR32	8	100	974	0.1027	410.18	23.72	727.53	94.80%	52
[8] Random Iteration	8	100	1022	0.0978	424.38	52.92	566.34	90.35%	84

Figure 1. Total Simulation Time for Each Algorithm

4.2. Total Completed Jobs

We ran the simulation for 1600ms for algorithms with 4 processors and 800ms for algorithms with 8 processors to see how many jobs could be completed.

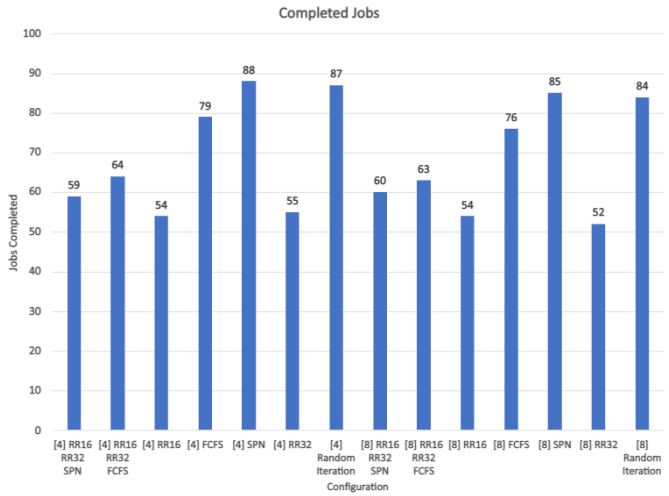

Figure 2. Total Completed Jobs

The Chart above shows the Total Completed Jobs:

- SPN and Random Iteration completed the highest number of jobs for simulations with 4 and 8 processors each.
- Similarly, RR16 and RR32 completed the lowest number of jobs for both 4 and 8 processors.

4.3. Average Wait Times

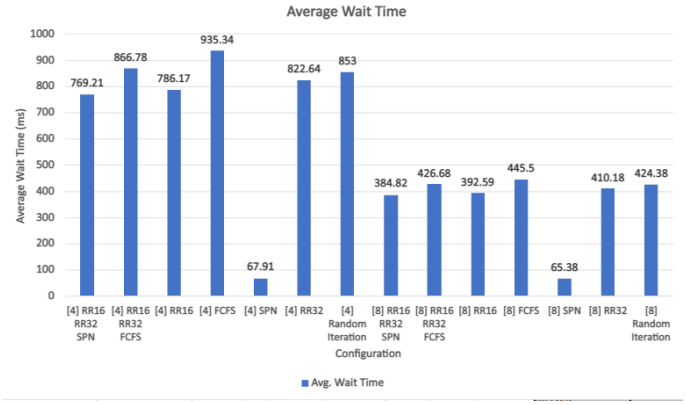

Figure 3. Average Wait Times for Each Algorithm

The Chart above shows the Average Wait Time Chart:

- The Algorithm with the worst WaitingTime is the First Come First Serve single-level queue, averaging 5-1500% more waiting time than other algorithms.
- The most efficient algorithm related to Waiting Time is the Shortest Processor Next algorithm.
- Other multilevel queues are not as efficient as RR1, RR2, and SPN but still perform somewhat better than the First Come First Serve single-level queue.

4.4. Average Turnaround Time

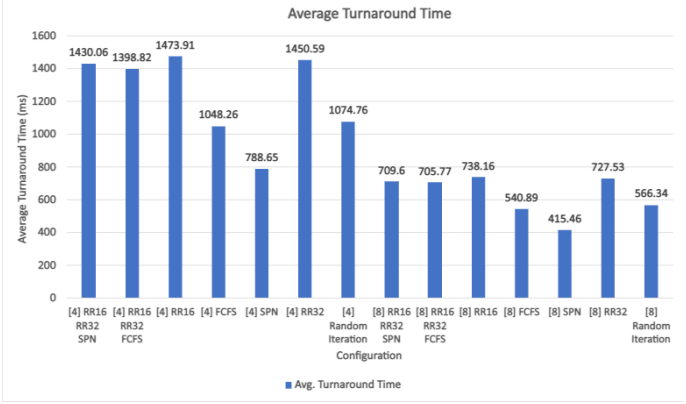

Figure 4. Average Turnaround Times for Each Algorithm

The chart below shows the Average Turnaround Time Chart:

- The Algorithm with the worst TurnAround time would be the RR16 algorithm, averaging 3-183% more TurnAround time than other algorithms.
- The most efficient algorithm related to Turnaround Time would be the Shortest Processor Next single queue.

4.5. Average Response Time

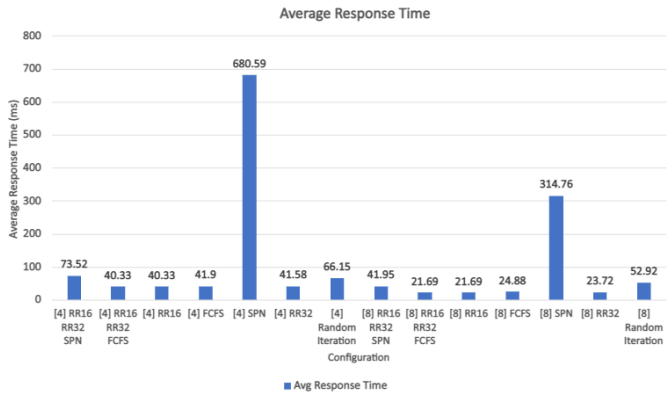

Figure 5. Average Response Times for Each Algorithm

The Chart below shows the Average Response Time Chart:

- The Algorithm with the worst Response Time would be the Shortest Processor Next Algorithm averaging 211-1831% more response time than others.
- The most efficient algorithm related to Response Time would be Round Robin 1, Round Robin 2, and First Come First Serve multilevel queues.
- Other multilevel queues are not as efficient as RR1, RR2, and FCFS but still perform at least 72% less Response Time compared to the Shortest Processor Next single queue algorithm.

4.6. Average Processor Utilization

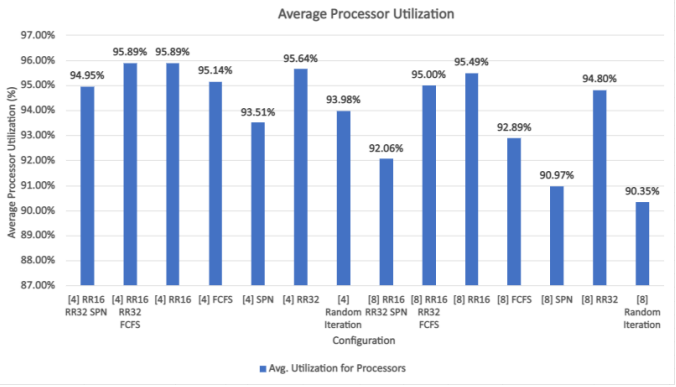

Figure 6. Average Processor Utilization for Each Algorithm

The Chart below shows the Average Utilization Percentage Chart:

- The Algorithm with the highest CPU Utilization time would be the RR1, RR2, and FCFS averaging 95.8% across 2 different processor variations tests.
- The lowest percentage of CPU Utilization Time is the First Come First Serve single-level queue, which utilizes way less compared to other algorithms.
- Other queue variations perform slightly less compared to RR1, RR2, and FCFS.

4.7. Analysis based on "An Efficient Randomized Algorithm for Real-Time Process Scheduling in PicOS Operating System"

The Chart below shows the Average Throughput between FCFS and Random Iteration:

- In comparison, Random Iteration multilevel queues managed to perform better than FCFS single-level queues when it comes to Average Throughput and Average Waiting Time.

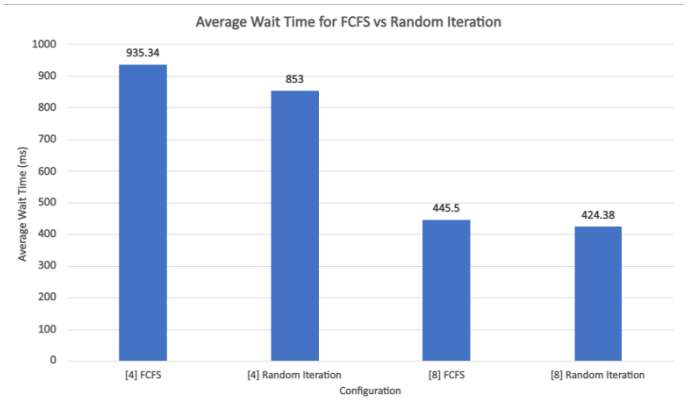

Figure 7. Average Wait Time for FCFS vs Random Iteration

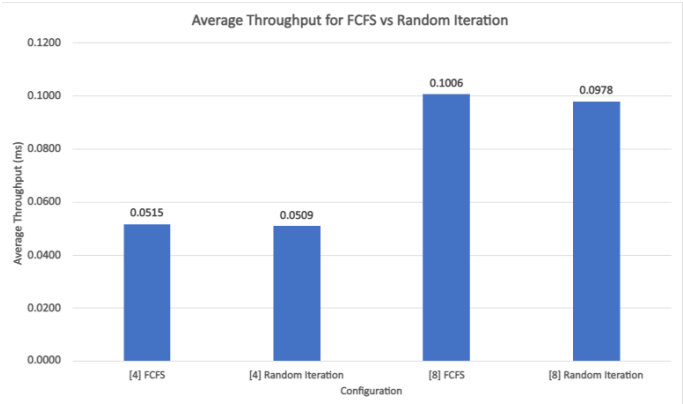

Figure 8. Average Throughput Time for FCFS vs Random Iteration

The simulations have proven Helmy's claims that Random Iteration Algorithms are very effective in many factors compared to First Come First Serve including Waiting Time, Throughput, and Response Time. [1]

5. Conclusion

Based on our analysis, the data suggests that the First-Come First-Serve (FCFS) has a higher average wait time (AWT). The Shortest Processor Next (SPN) algorithm is more efficient in terms of reducing wait times and turnaround times, whereas RR16, RR32, and FCFS multi-level queues demonstrate higher CPU utilization. Moreover, similar to the research, the random iteration algorithm is more effective in improving throughput, wait times, and response times compared to FCFS. SPN and Random Iteration have the most efficient algorithm in terms of completing tasks within a given time period. Thus, these points highlight the importance of selecting appropriate scheduling algorithms to optimize performance and resource utilization.

References

[1] Helmy*, T., Fatai, A., and Sallam, E.-S., "An Efficient Randomized Algorithm for Real-Time Process Scheduling in PicOS Operating System", in Advanced Techniques in Computing Sciences and Software Engineering, 2010, p. 117. doi:10.1007/978-90-481-3660-5-20.

[2] Stallings, William. Operating Systems : Internals and Design Principles. Upper Saddle River, N.J. :Prentice Hall, 2001.

[3] Chen, Shibo et al. "Deep Dive Into the Cost of Context Switch." (2019).

[4] Larsen, S., Lee, B. (2014). Survey on System I/O Hardware Transactions and Impact on Latency, Throughput, and Other Factors. Advances in Computers, 92, 67-104. doi:10.1016/B978-0-12-420232-0.00002-7