ISSN: 0067-2904

Optimal CPU Jobs Scheduling Method Based on Simulated Annealing Algorithm

Emad I Abdul kareem*, Salam Ayad Hussein
Department of Computer Science, Mustansiriyah University, Baghdad, Iraq

Received: 6/10/2021 Accepted: 9/1/2022 Published: 30/8/2022

Abstract

Task scheduling in an important element in a distributed system. It is vital how the jobs are correctly assigned for each computer's processor to improve performance. The presented approaches attempt to reduce the expense of optimizing the use of the CPU. These techniques mostly lack planning and in need to be comprehensive. To address this fault, a hybrid optimization scheduling technique is proposed for the hybridization of both First-Come First-Served (FCFS), and Shortest Job First (SJF). In addition, we propose to apply Simulated Annealing (SA) algorithm as an optimization technique to find optimal job's execution sequence considering both job's entrance time and job's execution time to balance them to reduce the job's waiting time to be executed. As a result, this research proves that the proposed technique achieves an optimization efficiency with a percentage average 45.5 % according to the FCFS algorithm and 54.5% according to SJF method.

Keywords: Operation System, CPU Tasks Scheduling, Artificial Intelligent, Meta-Heuristic Techniques, Optimization Techniques

طريقة مثلى لجدولة مهام وحدة المعالجة المركزية بألاعتماد على خوارزمية محاكاة التلدين

عماد عيسى عبد الكريم*, سلام اياد حسين
قسم علوم الحاسبات, كلية التربية, الجامعة المستنصرية, بغداد, العرا ق

الخلاصة

تعد جدولة المهام في نظام موزع أحد العناصر المهمة. من الأهمية بمكان كيفية تعيين الوظائف بشكل صحيح لكل معالج كمبيوتر لتحسين الأداء. تحاول الأساليب المقدمة تقليل النفقات وتحسين استخدام وحدة المعالجة المركزية. هذه التقنيات في الغالب تفتقر إلى التخطيط وبحاجة إلى أن تكون شاملة. لمعالجة هذا الخطأ ، تم اقتراح تقنية جدولة أمثلية هجينة للتهجين لكل من الخدمة المقدمة أولاً (FCFS) وأقصر مهمة أولاً (SJF). بالإضافة إلى ذلك ، سيُقترح تطبيق خوارزمية محاكاة التلدين (SA) كتقنية تحسين للعثور على تسلسل تنفيذ الوظيفة الأمثل مع الأخذ في الاعتبار وقت دخول الوظيفة ووقت تنفيذ المهمة لتحقيق التوازن بينهما لتقليل وقت انتظار الوظيفة ليتم تنفيذها. نتيجة لذلك ، أثبت هذا البحث أن التقنية المقترحة تحقق كفاءة التحسين بمتوسط نسبة 45.5٪ وفقًا لخوارزمية FCFS و 54.5٪ وفقًا لطريقة SJF.

*Email: mmimad72@uomustansiriyah.edu.iq

1. Introduction

In multi-processing systems, there are several programs that the user executes at the same time that contains several processes that require the CPU to finish its task; however, just one job will acquire the CPU at an any time. Therefore, CPU scheduling is required to form an economical and quicker system that permits a job to use the CPU whereas another one is on hold as a result of its awaiting different resources [1]. Most processes need to be executed; therefore, it is important to maximize the CPU utilization and output, as well as minimize turnaround waiting and response time. These criteria are achieved using CPU programming algorithms that manage how processes enter the CPU [2]. An example of those methods is the First Come First Served (FCFS) algorithm, which supplies the CPU to the first arrival. Also, the Shortest Job First (SJF) algorithm, that offers the central processing unit to the shortest process first [1]. To illustrate the methodology to be used to improve the performance of those algorithms, and the results to be reached, researchers have proposed numerous ways to enhance processor improvements' criteria through totally different algorithms to decrease the waiting time, increase response time, and speed up the turnaround time; however, there is no best algorithm for all criteria.

This paper concentrates on the hybrid optimization scheduling techniques. A hybrid optimization scheduling technique is proposed for hybridization of both First-Come First-Served (FCFS) and Shortest Job First (SJF). In addition, it will be applied to the Simulated Annealing (SA) algorithm as an optimization technique to find optimal job's waiting time by considering both job's arrival time and job's execution time to apply an efficient balance between them to reduce the job's waiting time.

1.1. CPU Scheduling

Processor scheduling is the process of allocating CPU time to processes or allocating CPU time slots to processes. This is a dynamic procedure that becomes more difficult in the event of a multiprocessing system. The main objectives of scheduling are [3]:

1) To be consistent across processes.
2) To increase the range of the processes that are done per unit time (called throughput).
3) To avoid postponing any procedure indefinitely (called Starvation Free).
4) To reduce "Overhead," which means wasteful time spent on scheduling.
5) To balance resource consumption, by using all resources at all times.
6) To implement a priority mechanism to give some processes extra CPU time.
7) To degrade elegantly in the face of high loads.

In the direction of relevance, there are many types of scheduling algorithms that have been presented in [4]. In the following sections, two schedules' methods are presented (the First Come First Served (FCFS) algorithm and the Shortest Job First (SJF) algorithm) which are under debate in this research.

1.2. First Come First Scheduling (FCFS)

The first come first served scheduling (FCFS) is the simplest scheduling algorithm. In this case, the process that first requests the CPU will be allocated the CPU first, where FIFO queue management strategy is used for FCFS implementation. FCFS is not a preemptive scheduling, which means if the CPU is assigned to a process, the process will occupy the CPU until it completes or an I/O is requested [5-7].

FCFS is problematic for time-sharing systems because not every process due to its inactivity can periodically share the CPU. Processes of this sort are sent to the "ready queue" depending on the arrival time. Once a process gets access to the CPU, it will run until it is finished. Because FIFO is non-preemptive scheduling, it may be utilized in "single programming" environments. It cannot be used as a master scheme in multiprogramming, but only as part of it. FCFS can also enclose the system in a dynamic system that is manned in another way known as a convoy effect. If a CPU-intensive process blocks the CPU, multiple I/O intensive

processes can be backed up, which leaves the I/O devices idle. This can also leave the CPU idle while everyone queues for I/O, and the cycle repeats itself when the CPU-intensive process comes back on the queue [8-10].

1.3. Shortest Job First (SJF)

Shortest Job First (SJF), also known as Shortest Job Next (SJN) or Shortest Process Next (SPN), is an inactive scheduling strategy that selects the waiting process with the shortest execution time to be executed next [10-12].

The shortest job first is advantageous because of its simplicity and it also minimizes the average amount of time each process has to wait until its execution is completed [13-15].

However, the disadvantage of SJF is that adding short processes can disrupt processes that take a long time to complete. Another disadvantage of using SJF is that the total execution time of a job must be known before execution, which is not possible [16,17].

It turns out that this problem is resolved with a very simple technique; in fact, it is a concept borrowed from operations research [C54, PV56] and applied to the scheduling of computer system duties. This new scheduling discipline is called the Shortest Task First (SJF) discipline, and the name should be easy to remember because it fully defines the policy of: the smallest job first, then the shortest, and so on. [18].

1.4. Simulated Annealing Algorithm

SA is a method of optimization that can be used to address a wide variety of issues. For complicated optimization issues, SA is recommended. SA Algorithm is shown in Figure 1, it starts at a fixed temperature, then the temperature gradually decreases as time passes, as shown in the Simulated Annealing Algorithm [19]. The solution is usually represented by a set of variables, but it can be defined by other means. Once the algorithm has begun, the solution gradually approaches the global minimum that probably occurs in a complex error surface. SA has been used in many fields due to its great robustness. One of SA's main features is that, while the solution might not be optimal, it still offers a solution. SA can provide a realistic alternative for solving certain optimization problems that cannot be easily modelled [20].

Simulated annealing algorithm

```
Select the best solution vector x0 to be optimized
Initialize the parameters: temperature T, Boltzmann's constant k, reduction factor c
while termination criterion is not satisfied do
    for number of new solution
        Select a new solution: x0+Δx
            if f(x0+Δx) > f(x0) then
                fnew = f(x0 + Δx);   x0 = x0 + Δx
                else
                    Δf = f(x0 + Δx) − f(x0)
                    random r(0, 1)
                        if r > exp(−Δf/kT) then
                            fnew = f(x0 + Δx),   x0 = x0 + Δx
                            else
                            fnew = f(x0),
                        end if
            end if
        f = fnew
        Decrease the temperature periodically: T = c x T
    end for
end while
```

Figure -1 Pseudo-Code for Simulated Annealing Algorithm [21].

The law of thermodynamics state that at a temperature t, the probability of an increment in energy Δf, has been given by equation (1).

$$P(\Delta f) = exp\ (-\Delta f /kt) \quad \ldots\ldots\ldots (1)$$

Where k is Boltzmann's constant.

The simulation in the Metropolis algorithm calculates the new energy of the system. If the energy has increased, then the new state is accepted using the probability by the equation (1). Otherwise, the system will keep the current state and reject the new state. At each iteration, a certain number is carried out at a temperature, then the temperature is decreased. This is repeated until the system cools into a steady state. This equation is used in simulated annealing, where the Boltzmann constant is ignored. Therefore, the probability of accepting a worse state is given by the equation 2.

$$P = exp\ (-\Delta fT) > r \quad \ldots\ldots\ldots (2)$$

Where Δf equals the change in the evaluation function T, which equals the current temperature r = a random number between 0 and 1. The probability of a wrong decision depends on the temperature of the system and the changes in the waiting time function. It can be seen that the lower the temperature of the system is, the lower the possibility of accepting the worst motion. This is like gradually moving to a cool state in physical annealing. Note that when the temperature is zero, only the better moves will be accepted, which effectively makes simulated annealing act [22]. The essence of the metropolis test lies in the following three principles:

(i) The chance of acceptance increases with high-temperature values, facilitating transitions of design to a thorough early exploration of the design space.

(ii) Low Δf, values are more promising, which results in greater acceptance than big values.

(iii) As temperature decreases, the chance of acceptance reduces substantially; therefore, ensuring more useful searching in subsequent phases.

After all internal loop iterations have been finished at a certain temperature level, the temperature for the following cooling cycle has been determined by the multiplication of the cooling factor of the current temperature. The aforementioned method is performed until it finished the iterations of all cooling cycles [23].

2. RELATED WORKS

In [6], Fawad Ahmad et al. showed that the proposed hybrid scheduling discipline is more efficient and enhances the absence of existing scheduling. It helps people using multi-programming environment and studying an operating system. As a consequence of designing a unique priority task policy, identical scalability tests were achieved in [7]. In addition to controlling the significance of mixed hard real-time and soft time jobs in the system.

The new improved Round Robin (RR) technology; which is supported by either one processor or multi-processor system, has been developed by Shihab Ullah [24]. The main aims of this approach were to minimize the average duration of waiting and turning while maximizing the overall output from context-switching between different jobs. Priyanka Sangwan1et al in [25], compared the original RR with a suggested model known as a standard cloud computing resource scheduling method, which increases loading balance and shows outcomes in the cloud.

Vaishali Chahar & Supriya Raheja in [26], proposed a new multi-level queue-based CPU scheduling system. The suggested approach allocates the service time for the CPU and dynamically determines the value of time quantity (TQ) among the multiple queues. The scheduling approach used in the MathLab application is implemented by the "fuzzy toolbox". Simulation research termed Markov-chain analyses was given by Shweta Jaine & Saurabh

Jain [27] to identify the influence of the waiting state, with overall system efficiency and throughput on the scheduling approach of "multi-level feedback queue".

This research also emphasizes that the comparative analysis is estimated in an arithmetic model utilizing changing α and d values. A. Maktum et. al. in [28] proposed an idea to discover nearly optimum solutions by using a "genetic algorithm" for the problem of CPU scheduling. They have created a simple scheduling algorithm based on their evolutionary method for uniprocessor scheduling and compared their scheduling algorithm with SJF and FCFS scheduling to minimize average wait times.

Shatha Jawad, in [9] proposed a Neuro-fuzzy scheduling approach for the application and amended algorithm of CPU scheduling to maximize reaction time and minimize average time and turnaround time. This is done by integrating well-known timetables, including SJF preventive scheduling and scheduling using the neuro-fuzzy methodology.

Kumar Saroy, Sushil and others in [29] proposed an approach which was simulated and implemented in C++ programming. This method addressed various problems such as extended average waiting times, turnover times, indefinite hunger blockage, and practical execution. The proposed scheduling eliminated the numerous issues and it is easy to implement with two slice time types. In [30], Omar Ahmed and Adnan Brifcani profound training used a multi-layered artificial neural network for learning advanced characteristics, such as deep neural and neural networks. The key characteristics gained from data were used for deep learning.

Elmougy, and others in [31] proposed a novel hybrid task scheduling algorithm named (SRDQ) combining Shortest Job First (SJF) and Round Robin (RR) schedulers considering a dynamic variable task quantum. The proposed algorithms mainly rely on two basic keys; the first having a dynamic task quantum to balance waiting time between short and long tasks while the second involves splitting the ready queue into two sub-queues, Q1 for the short tasks and the other for the long ones. Assigning tasks to resources from Q1 or Q2 are done mutually two tasks from Q1 and one task from Q2.

Sai, R Vijay and others in [32] proposed a hybrid algorithm to find an answer for the scheduling problem. This algorithm combines shortest job first and longest job first scheduling algorithm and based on the conclusion, it is proposed whether it is suitable in uniprocessor as well as in multiprocessor environment. Given other parameters like waiting time and turnaround time, this algorithm can achieve efficient solutions.

Puneet Himthani and others in [33] proposed a Multi-Tasking Scheduling Scheme that optimizes the system's throughput and reduces the overheads caused by context switches. The proposed scheme is a hybrid of Round Robin and Shortest Remaining Time First approaches as it encompasses the advantages of both methods. In this scheme, Slice Bit and remaining Burst Time decide the update of time quantum. However, the proposed scheme may not be as fair as traditional Round Robin.

The previous works have been examined in pursuit of developing the proposed method in this study to avoid their drawbacks. Their parameters, ideas and best features are the inspiration to the proposed technique of this paper.

3. PROPOSED METHOD

The proposed method considers a finite number of jobs with their corresponding *jobs' arrival time* and the *job's execution time*. These jobs are placed in a job queue from which jobs are assigned to be implemented by an operating system. The proposed method will concentrate on the *job's execution time* and the *job's arrival time* via taking advantage of the characteristics of (FCFS) and (SJF) methods. Therefore, this trend will produce a scheduling method that strikes a balance between these two methods. Through this balance, the waiting time will be reduced compared to these two methods. The proposed algorithm has been illustrated in Figure 2.

Algorithm of Optimal CPU Tasks Scheduling Method Based on Simulated Annealing Algorithm
Input: ***Queue-Jobs:*** Array [1...N] where: **N** is the number of Jobs. {Note: In this queue, the index number of this queue represents the job entrance sequence in the queue which represents the job's arrival time, where the element's value of the queue represents the job's execution time. }
Output: ***Solution***: Array [1...N] where: **N** is the number of Jobs. {Note: In this queue, the index number of this queue represents the job entrance sequence in the queue which represents the job's arrival time, where the element's value of the queue represents the job's execution time. }
Step 1: *Solution= Queue-Jobs.* Step 2: Let $Old\_Optimal\ Waiting\ Time = \left\lvert \sum_{i=1}^{N} Solution_i + i + Panalty_i \right\rvert$. Step 3: Let T=1.0 {give the Temperature value T its initial value 1.0}. Step 4: Let T_min =0.00001 {give the minimum temperature value its initial value 0,00001}. Step 5: Let Alpha=0.9 {give the Alpha value its initial value 0.9 which it will control the temperature decreasing process} Step 6: Repeat Step 6.1 to Step 6.3 until T< T-min: Step 6.1: Let i=1 Step 6.2: Repeat Step 6.2.1 to Step 6.2.5 until i<=N: Step 6.2.1: *New_Solution* = Neighbor (Solution) {It Will be Created Randomly} Step 6.2.2: $New\_Optimal\ Waiting\ Time = \left\lvert \sum_{i=1}^{N} New\_Solution_i + i + Panalty_i \right\rvert$ Step 6.2.3: $Acceptance\_Probability = e^{\frac{Old\_Optimal\ waiting\ time - New\_optimal\ waiting\ time}{T}}$ Step 6.2.4: If *Acceptance_Probability> Random_Number* then *Soluation=New_Soluation* $Old\_waiting\_time = New\_waiting\_time$ End If Step 6.2.5: Let *i=i*+1 Step 6.3: Let *T=T*Alpha* Step 7: End.

Figure - 2 Algorithm of Optimal CPU Tasks Scheduling Method Based on Simulated Annealing Algorithm.

In Figure 2, the input of the algorithm is a set of *N* jobs. On the other hand, the output will be the same set of *N* jobs, but after arranging them in a queue according to their sequence (which means their role) in implementation within the operating system. In these two queues, the index number of these queues represents the job entrance sequence in the queue which represents the *job's arrival time*, where the element's value of the queue represents the *job's execution time*. (See Figure 3).

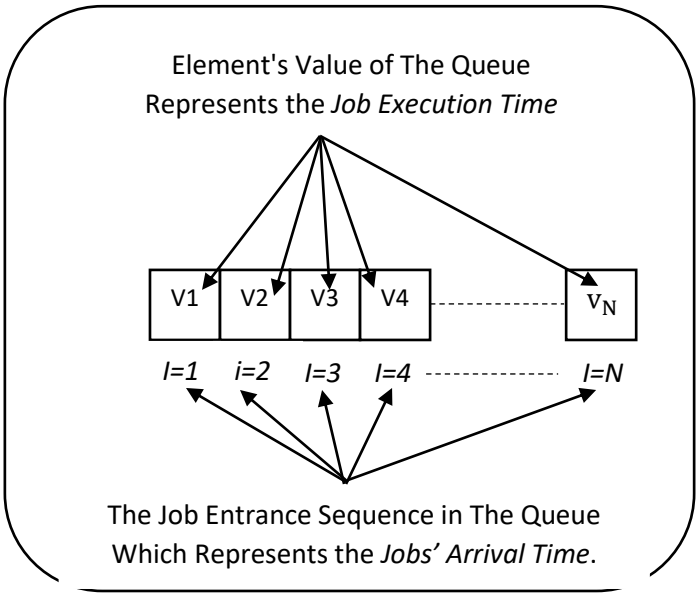

Figure 3-The Structure of the queue-jobs and solution queue.

Based on the SA algorithm; initially, these jobs are randomly arranged. After the jobs are randomly arranged, they will be initially considered as an initial set of jobs. Therefore, the total old and the new *Optimal Waiting Time* will be calculated for every job in the *Queue-Jobs* of each test based on three criteria; the *job's execution time* (which represent the element's value of the queue), the *job's arrival time* (which represent The Job Entrance Sequence in The Queue), and the *penalty* that will be imposed on the job based on the job's arrival time (see equation 2).

$$Optimal\ Waiting\ Time = \left|\sum_{i=1}^{N} Solution_i + i + Panalty_i\right| \quad \text{..... (2)}$$

In equation 1, $Solution_i$ represents the *job's execution time*, where i represents the Job Entrance Sequence in The *Queue Jobs*. Finally, $Panalty_i$ represents the penalty value that is added to the total value of each job's waiting time. The penalty value of each job will be assigned starting from 0 for the first job (Job with sequence 1) considering that no penalty is given to the first job in the queue because it is the first job entered into the queue. The value of the assigned penalty will gradually increase whenever the job's arrival time is late. The role of the penalty is to provide a balance between the characteristics of the two methods First-Come First-Served (FCFS), and Shortest Job First (SJF) in terms of choosing which job will be implemented first by the operating system.

It also starts with an initial temperature T = 1.0. The temperature T will be reduced at the end of each iteration and before the next iteration by using *Temperature Reduction Function Alpha*, where the initial value of *Alpha* is 0.9 to control the temperature decreasing process as it is shown in equation 3.

$$T = T * Alpha \quad \text{.....(3)}$$

Starting at the initial temperature value, the processes will be looped through decreasing the temperature according to alpha. These loops will be stopped when the termination condition is reached. The termination condition is the acceptable temperature threshold (Until T < T-min).

For each iteration, the neighbourhood of solutions pick one of the *New_Solution* and calculate the *Acceptance_Probability* for the *New_Solution* using equation 4. Then according to a *Random_Number* value which is created for this purpose, the *New_Solution* will be accepted if the *Acceptance_Probability* is greater than the *Random_Number* value. Otherwise, the

New_solution should be rejected. Thus, the *New_Solution* will be accepted if it has a shorter waiting time compared to the *Old_Soluation*.

$$Acceptance\_Probability = e^{\frac{Old\_optimal\ waiting\ time - New\_optimal\ waiting\ time}{T}} \quad \ldots\ldots\ldots (4)$$

4. RESULTS AND ANALYSIS

The results were analyzed via quantitative and qualitative evaluations. One hundred tests have been used in this analysis. Each test has 100 random jobs, the randomness come from assigning a random job's arrival *time* as well as a random *job's execution time* for the 100 jobs of each test. The proposed schedule algorithm was evaluated by finding the *Optimization Ratio* of the job's waiting time gained via using the proposed scheduling algorithm for each test. The *Optimization Ratio* of the job's waiting time reflects the ability of the proposed algorithm in terms of reducing the total jobs waiting time for each test. Thus, the proposed scheduling algorithm should be an optimization algorithm that balance the two mentioned criteria's, which are job's arrival time and job's execution time. The *Optimization Ratio OR* for the job's waiting time was calculated via equation 5 where, the average of 100 jobs per each test was calculated via equation 6.

$$OR = \frac{Fw}{Iw} \times 100\% \quad \ldots\ldots\ldots\ldots\ldots\ldots (5)$$

Where:

OR is the *Optimization Ratio* for the job's waiting time for each job in each test.
Fw is the *Optimal Waiting Time* of each job in the final solution.
Iw is the *Optimal Waiting Time* of each job in the initial solution.

$$OR_{average} = \frac{\sum_{i=1}^{100} OR_i}{100} \quad \ldots\ldots\ldots\ldots\ldots (6)$$

Where:

$OR_{average}$ is the average optimization ratio of the Job's waiting time for each test.

In equation 5, both *Fw* and *Iw* was calculated using equation 2. Equation 5 calculates the *Optimization Ratio OR* for the job's waiting time for the random initial solution and the final solution resulted one from the proposed scheduling algorithm. Finally, equation 6 calculated the *Average Optimization Ratio* for the Job's waiting time of each job of the 100 tests (see Figure 4).

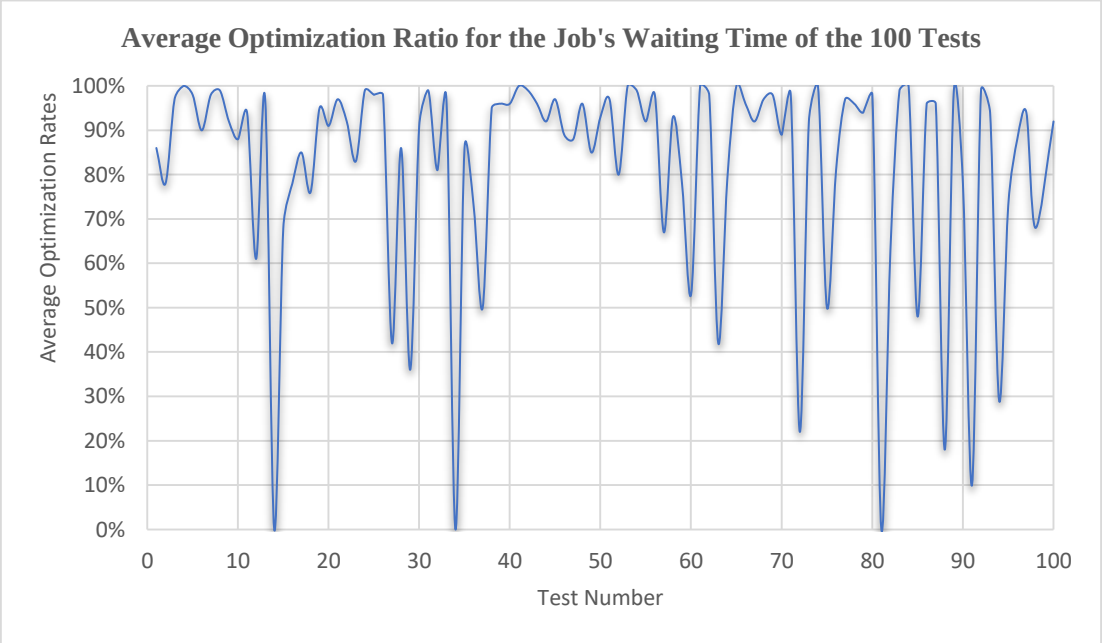

Figure 4-The Average Optimization Ratio for Job's Waiting time of the 100 Tests.

As shown in Figure 4, it can be noticed that from the 100 tests, 59 tests have a high optimization ratio for job's waiting time (90%-100%). While 14 tests with *Optimization Ratio* for job's waiting time in the range of 80%-90%. In addition, the *Optimization Ratio* for the job's waiting time for the rest 27 tests is less than 80%.

Also in Figure 4, the *Optimization Ratio* of the job's waiting time for the mentioned 27 tests is relatively small because the random job's sequence in which the proposed schedule algorithm initially starts with a waiting time ratio close to or equal to the optimal value of the job's waiting time ratio. Thus, the remaining optimization ratio of the job's waiting time needs to reach the highest value was small or none, before the proposed schedule algorithm needed to stop it's work due to one of the termination conditions. This explains the low optimization rates of the job's waiting time that appeared in Figures 4 and 5.

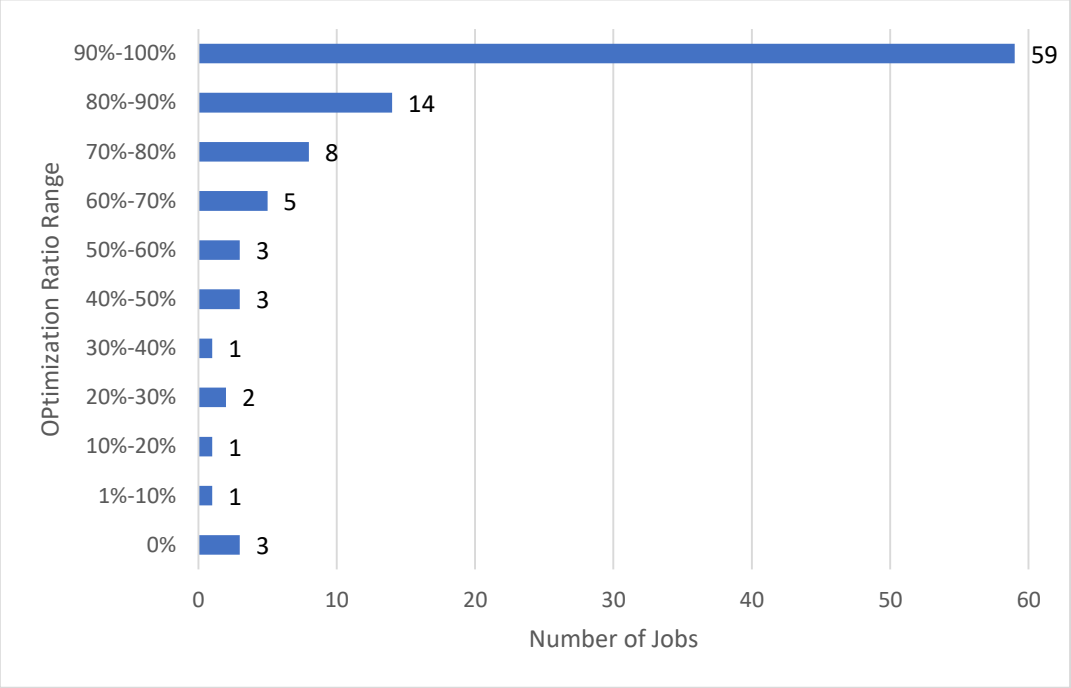

Figure 5-Frequency Distribution for Optimization Ratio of the 100 Tests.

5. QUANTITATIVE AND QUALITATIVE COMPARISONS

The comparisons made in this section will depend on the percentage at which the waiting time is reduced for each job in each test. The number of tests was 100. The waiting time for a specific job will be calculated based on the sum of the execution time for all the jobs that precede it (see equation 4).

$$Waiting\ Time_i = \sum_{j=1}^{i-1} Execution\ Time_j \dots\dots\dots (4)$$

Where: $i > 1$ because there are no jobs to wait for by the first job.

$Waiting\ Time_i$ is the waiting time that is needed to execute the ith job.

$Execution\ Time_j$ is the execution time of the jth job.

To evaluate the capabilities of the proposed algorithm to reduce the waiting time of the jobs, the 100 tests have been compared with FCFS and SJF methods according to the percentage of reducing the waiting time by the proposed algorithm. It is clear that the proposed algorithm has a positive effect on reducing the waiting time. Since the percentage of reducing the

average waiting time for jobs compared with the FCFS, and SIJ algorithms are 45.5%, and 54.5% respectively (see Figure 6). That means, the proposed algorithm achieved its main goal, which is reducing the waiting time of the jobs by balancing among the characteristic of the FCFS and SJF. It is worth mentioning that these jobs are standing in the queue of jobs to be implemented by the operating system.

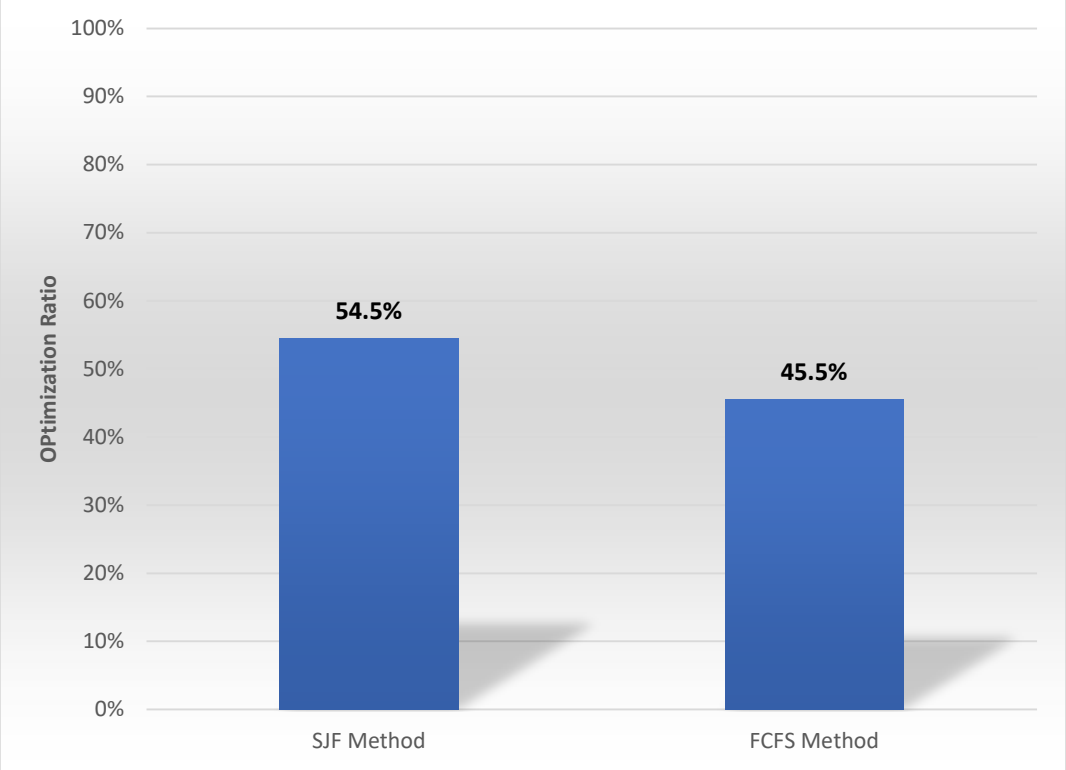

Figure 6-The Average of Waiting Time Reduction of The Proposed Method Comparing with Both FCFS and SJF Methods.

6. CONCLUSION

In this paper, the technique of combining the advantages of the FCFS algorithms and the SJF algorithm was used. Thus, the proposed algorithm become a balancing algorithm between the above two algorithms. The proposed algorithm has the advantage of executing the job according to the time it enters the queue in conjunction with the feature of executing the task with the shortest jobs. Accordingly, there is a clear reduction in the average waiting time by 45.5% according to the FCFS algorithm and 54.5% according to the SJF algorithm. Thus, it is indicating that the proposed algorithm successfully balanced between both algorithms and achieving the shortest waiting time for jobs. It is concluded from this, that the proposed algorithm achieves the main research's goal for which this algorithm has been developed.

Acknowledgements

This research was supported by the Computer Science Department/ College of Education/ Mustansiriyah University. We strongly express our gratitude to the Department of Computer Science for the continuous and diligent support for us in this work.

References

[1] Peter Baer Galvin, Greg Gagne, Silberschantz A. Wiley: *Operating System Concepts, 9th Edition - Abraham Silberschatz, Peter B. Galvin, Greg Gagne* 2005.

[2] Hutagalung A. MULTIPROCESSOR AND REAL-TIME SCHEDULING SHORTEST-JOB-FIRST (SJF) SCHEDULING ALGORITHM Adel. Angew Chemie Int Ed 6(11), 951–952 1967;3:5–24.

[3] Englander I. The Architecture of Computer Hardware and Systems Software;An Information Technology Approach. 3rd Editio. John Wiley & Sons, Inc.,; 2003.
[4] Magdalene, R., & Sridharan D. Comparative Analysis of FCFS and SJF for Multimedia Process Scheduling. Adv Commun Syst Networks 2020;7:667–72.
[5] Sowmya G, Chinaappalanaidu R. A COMPARISON OF SCHEDULING ALGORITHM FOR BEST UTILIZATION OF MEMORY 2018;120:3563–70.
[6] Akhtar M, Hamid B, Humayun M. An Optimized Shortest job first Scheduling Algorithm for CPU Scheduling. J Appl Environ Biol Sci 2015;5:42–6.
[7] Harki N, Ahmed A, Haji L. CPU Scheduling Techniques: A Review on Novel Approaches Strategy and Performance Assessment. J Appl Sci Technol Trends 2020;1:48–55. https://doi.org/10.38094/jastt1215.
[8] Krishna MV. BigData Processing using First Come First Served (FCFS) Algorithm 2018;7:83–7.
[9] Jawad S. Design and evaluation of a neurofuzzy CPU scheduling algorithm. Proc 11th IEEE Int Conf Networking, Sens Control ICNSC 2014 2014:445–50. https://doi.org/10.1109/ICNSC.2014.6819667.
[10] Chauhan, H., & Inani A. Modified Concept to Achieve Maximum Efficiency of CPU Scheduling Algorithm. 3rd Int. Conf. Electron. Commun. Aerosp. Technol. (ICECA), IEEE, 2019, p. 660–3.
[11] Kumar S, Kumar G, Jain K, Jain A. An approach to reduce turn around timeand waiting timeby the selection of round robin and shortest job first algorithm. Int J Eng Technol 2018;7:667. https://doi.org/10.14419/ijet.v7i2.8.10553.
[12] Chandra Shekar N, Karthik V. Analysis of Priority Scheduling Algorithm on the Basis of FCFS & SJF for Similar Priority Jobs. Int J Eng Res Comput Sci Eng 2017;4:73–6.
[13] Teraiya J, Shah A. Comparative Study of LST and SJF Scheduling Algorithm in Soft Real-Time System with its Implementation and Analysis. 2018 Int Conf Adv Comput Commun Informatics, ICACCI 2018 2018:706–11. https://doi.org/10.1109/ICACCI.2018.8554483.
[14] Garg S, Kumar D. A K-Factor CPU Scheduling Algorithm. 2018 9th Int Conf Comput Commun Netw Technol ICCCNT 2018 2018:1–6. https://doi.org/10.1109/ICCCNT.2018.8493662.
[15] Goel N, Garg RB. A Comparative Study of CPU Scheduling Algorithms. Int J Graph Image Process 2012;2:245–51.
[16] Singh P, Singh V, Pandey A, Robin R. Analysis and Comparison of CPU Scheduling Algorithms. Int J Emerg Technol Adv Eng 2014;4:91–5.
[17] Dash AR, Sahu S kumar, Samantra SK. An Optimized Round Robin CPU Scheduling Algorithm with Dynamic Time Quantum. Int J Comput Sci Eng Inf Technol 2015;5:07–26. https://doi.org/10.5121/ijcseit.2015.5102.
[18] Putra TD. Analysis of Preemptive Shortest Job First (SJF) Algorithm in CPU Scheduling. Ijarcce 2020;9:41–5. https://doi.org/10.17148/ijarcce.2020.9408.
[19] ELLIOTT SJ. 9 - Optimisation of Transducer Location. In: ELLIOTT SJ, editor. Signal Process. Act. Control, London: Academic Press; 2001, p. 411–38. https://doi.org/https://doi.org/10.1016/B978-012237085-4/50011-9.
[20] Sattar RA, Kareem EIA. Intelligent Dietician System. Solid State Technol 2020;63:3639–55.
[21] Rere LMR, Fanany MI, Arymurthy AM. Simulated Annealing Algorithm for Deep Learning. Procedia Comput Sci 2015;72:137–44. https://doi.org/10.1016/j.procs.2015.12.114.
[22] Henderson D, Jacobson SH, Johnson AW. The Theory and Practice of Simulated Annealing. 2006. https://doi.org/10.1007/0-306-48056-5_10.
[23] Kareem EIA, Safar FY. Intelligent Recommendation Module for Emergency Vehicles. Int J Traffic Transp Eng 2018;7:63–9. https://doi.org/10.5923/j.ijtte.20180703.03.
[24] Zhang C, Luo P, Zhao Y, Ren J. An efficient round robin task scheduling algorithm based on a dynamic quantum time. Int J Circuits, Syst Signal Process 2019;13:197–204.
[25] Alhaidari F, Balharith TZ. Enhanced round-robin algorithm in the cloud computing environment for optimal task scheduling. Computers 2021;10. https://doi.org/10.3390/computers10050063.
[26] Chahar V, Raheja S. Fuzzy based multilevel queue scheduling algorithm. Proc 2013 Int Conf Adv Comput Commun Informatics, ICACCI 2013 2013:115–20. https://doi.org/10.1109/ICACCI.2013.6637156.

[27] Jain DSJS. Analysis of Multi Level Feedback Queue Scheduling Using Markov Chain Model with Data Model Approach. Int J Adv Netw Appl 2016;07:2915–2924.

[28] Maktum TA, Dhumal RA, Ragha L. A genetic approach for processor scheduling. Int Conf Recent Adv Innov Eng ICRAIE 2014 2014:9–12. https://doi.org/10.1109/ICRAIE.2014.6909108.

[29] Kumarsaroj S, Sharma AK, Chauhan SK. A novel CPU scheduling with variable time quantum based on mean difference of burst time. Proceeding - IEEE Int Conf Comput Commun Autom ICCCA 2016 2017:1342–7. https://doi.org/10.1109/CCAA.2016.7813986.

[30] Magis AT, Funk CC, Price ND. SNAPR: A Bioinformatics Pipeline for Efficient and Accurate RNA-Seq Alignment and Analysis. IEEE Life Sci Lett 2015;1:22–5. https://doi.org/10.1109/lls.2015.2465870.

[31] Elmougy S, Sarhan S, Joundy M. A novel hybrid of Shortest job first and round Robin with dynamic variable quantum time task scheduling technique. J Cloud Comput 2017;6. https://doi.org/10.1186/s13677-017-0085-0.

[32] Sai RV, Lavanya M, Srinivasan B. A HYBRID ALGORITHM FOR MULTIPROCESSOR SCHEDULING 2018;118:3149–55.

[33] Himthani P, Mishra NK, Pare T, Dubey GP. Hybrid Multi-Tasking Scheduling Scheme based on Dynamic Time Quantum using Slice Bit for improving CPU Throughput. SSRN Electron J 2021. https://doi.org/10.2139/ssrn.3916291.