Journal of Theoretical and Applied Information Technology
15th March 2012. Vol. 37 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

THE NEW METHOD OF ADAPTIVE CPU SCHEDULING USING FONSECA AND FLEMING'S GENETIC ALGORITHM

[1]MEHDI NESHAT, [2]MEHDI SARGOLZAEI, [3]ADEL NAJARAN, [4]ALI ADELI

[1] Department of Software Engineering, Shirvan Branch, Islamic Azad University, Shirvan, Iran
[2] Department of Computer sciences, Shirvan Branch, Islamic Azad University, Shirvan, Iran
[3] Department of Software Engineering, Shirvan Branch, Islamic Azad University, Shirvan, Iran
[4] Department of Software Engineering, Shirvan Branch, Islamic Azad University, Shirvan, Iran

E-mail: [1]neshat_mehdi@ieee.org, [2]Sargolzaei@iau-shirvan.ac.ir , [3]nadjaran@iau-shirvan.ac.ir , [4]adeli@iau-shirvan.ac.ir

ABSTRACT

The CPU scheduling is one of the most important tasks of the operating system. Many algorithms are designed and used in this regard each having advantages and disadvantages. In this paper a new algorithm for the CPU scheduling is presented using FFGA (Fonseca and Fleming's Genetic Algorithm) multiobjective optimization. Contrary to the classical algorithms in use, it uses the three parameters of CPU burst time; I/O devices service time, and priority of process instead of using one parameter of CPU burst time. The important point is the adaptation of the algorithm which selects a special process depending on the system situation. The performance of this algorithm was compared with seven classical scheduling algorithms (FCFS, RR (equal, prioritized), SJF (preemptive, non-preemptive, Priority (preemptive, non-preemptive)), and the results showed that the performance of the proposed method is more optimized than other methods. The proposed algorithm optimizes the average waiting time and response time for the processes.

Keywords: *CPU Scheduling, Multiobjective Optimization, FFGA, Waiting Time, Response Time, Turnaround Time.*

1. INTRODUCTION

Each process needs two factors when entering the operating system; first CPU and second I/O. The processes are divided into two general groups of CPU limited and I/O limited processes based on the need to these two factors. However, most processes always need the two types of sources. Now in a system with many processes available there will be a competition between these processes to acquire the resources. One of the most difficult problems in designing the operating systems is the timely allocation of resources to the processes and retrieving them. This problem is solved by the classical view of the different algorithms. All of the algorithms have some advantages and disadvantages; yet there is not a general method available. This is more problematic in systems with multiprocessor.

Modern Operating Systems are moving towards multitasking environments which mainly depends on the PU scheduling algorithm since the CPU is the most effective or essential part of the computer. Round Robin is considered the most widely used scheduling algorithm in CPU scheduling [1]-[2], also used for flow passing scheduling through a network device [3]-[4]. Operating systems may feature up to 3 distinct types of a long-term scheduler (also known as an admission scheduler or high-level scheduler), a mid-term or medium term scheduler and a short-term scheduler. The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler [1].

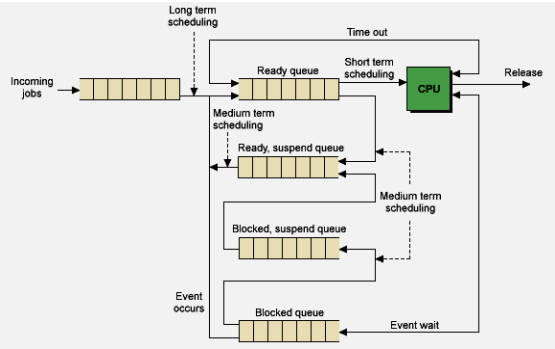

***Figure 1**: Queuing diagram for scheduling*

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The key idea behind a medium term scheduler is that some times it can be advantageous to remove processes from memory and thus reduce degree of multiprogramming. Later, the processes can be reintroduced into memory, and its execution can be continued where it left off. This scheme is called as swapping. So, the process is swapped out, and is later swapped in, by the medium term scheduler. The time for which a process holds the CPU is known as burst time. The time at which a process arrives is its arrival time. Turnaround time is the amount of time to execute a particular process. Waiting time is the amount of time a process has been waiting in the ready queue [5].

Using Artificial Intelligence methods in CPU scheduling algorithms has optimized the results. In recent years good scheduling algorithms have been presented which used tools like Fuzzy Logic[7-11], Neural network[12]-[18], Genetic Algorithms[13 - 14] and Swarm Intelligence[15]-[16]. In this study a multiobjective optimization method is used.

Multiobjective Optimization Problems (MOPs) optimize a set of conflicting objectives simultaneously. Mops are a very important research topic, not only because of the multi-objective nature of most real-world decision problems, but also because there are still many open questions in this area. In fact, there is no one universally accepted definition of optimum in MOP as opposed to single objective optimization problems, which makes it difficult to even compare results of one method to another. Normally, the decision about what the best answer is corresponds to the so-called human decision maker [19].

We will proceed to the main parameters of scheduling in the second section. The FFGA algorithm will be explained in the third section and the proposed algorithm will be reviewed in the forth section. In the fifth section the experimental results gathered through the proposed method will be compared with those of other methods and finally a general conclusion will be drawn.

2. METHODS

2.1. CPU SCHEDULING

Whenever the CPU becomes idle, the operating system must select one of the processes in the queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.

Note that the ready queue is not necessarily a first-in, first-out (FIFO) queue. As we shall see when we consider the various scheduling algorithms, a ready queue can be implemented as a FIFO queue, a priority queue, a tree, or simply an unordered linked list. Conceptually, however, all the processes in the ready queue are lined up waiting for a change to run on the CPU. The records in the queue are generally process control blocks (PCB) of the processes [1].

2.1.1. CPU Utilization

We want to keep the CPU as busy as possible that means CPU is not free during the execution of processes. Conceptually the CPU utilization can range from 0 to 100 percent.

2.1.1.1. Throughput

If the CPU is executing processes, then work is being completed. One measure work is the number of processes that are completed per time unit that means the number of tasks per second which the scheduler manages to complete the tasks.

2.1.1.2. Response Time

In an interactive system, turnaround time may not be best measure. Often, a process can produce some output fairly early and can continue computing new results while previous results are being output to the user. Thus, response time is the time from the submission of a request until the first response is produced that means when the task is submitted until the first response is received. So the response time should be low for best scheduling.

2.1.1.3. Turnaround Time

Turnaround time refers to the total time which is spend to complete the process and is how long it takes the time to execute that process. The time interval from the time of submission of a process to the time of completion is the turnaround time. Total turnaround time is calculation is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU and doing I/O.

2.1.1.4 Waiting Time

The waiting time is not the measurement of time when a process executes or does I/O completion; it affects only the amount of time of submission of a process spends waiting in the ready queue. So the Waiting time is the period of spent waiting in the ready queue to submit the new arriving process for the CPU.

So a good scheduling algorithm for real time and time sharing system are concluded that must possess following characteristics [6]:

1. Minimum context switches.
2. Maximum CPU utilization.
3. Maximum throughput.
4. Minimum turnaround time.
5. Minimum waiting time.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

6. Minimum response time.

2.2. Algorithms Scheduling

The different methods are used to allocate CPU to ready processes in queue. Each of these methods uses a certain algorithm and has advantages and disadvantages. Several common methods for process scheduling will follow.

2.2.1. First-Come, First-Served (FCFS):

This algorithm allocates the CPU to the process that requests the CPU first. This algorithm is easily managed with a FIFO queue. New process enters the queue through the tail of the queue and leaves through the head of the queue (when the process is allocated to the CPU) (1). The processes are allocated to the CPU on the basis of their arrival at the queue. Once a process is allocated to the CPU, it is removed from the queue. A process does not give up CPU until it either term unites or perform s I/O [1].

2.2.2. Shortest-Job-First (SJF):

The SJF algorithm associates the lengths of the next CPU burst w ith each process such that that the process that have the smallest next CPU burst is allocated to the CPU (1). The SJF uses the FCFS to break tie (a situation where two processes have the same length next CPU burst). The SJF algorithm may be implemented as either a preemptive or non-preemptive algorithms. When the execution of a process that is currently running is interrupted in order to give the CPU to a new process with a shorter next CPU burst, it is called a preemptive SJF. On the other hand, the non-preemptive SJF will allow the currently running process to finish its CPU burst before a new process is allocated to the CPU [1].

2.2.3. Priority Scheduling (PS):

The PS algorithm associates with each process a priority and the CPU is allocated to the process based on their priorities. Usually, lower numbers are used to represent higher priorities. The process with the highest priority is allocated first. If there are multiple processes with same priority, typically the FCFS is used to break tie [1].

2.2.4. Round Robin (RR):

The RR algorithm is designed especially for time-sharing systems and is similar to the FCFS algorithm. Here, a small unit of time (called time quantum or time slice) is defined. A time quantum is generally from 10-100 milliseconds. So, the RR algorithm will allow the first process in the queue to run until it expires its quantum (i.e. runs for as long as the time quantum), then run the next process in the queue for the duration of the same time quantum. The RR keeps the ready processes as a FIFO queue. So, new processes are added to the tail of the queue. Depending on the time quantum and the CPU burst requirement of each process, a process may need less than or more than a time quantum to execute on the CPU. In a situation where the process need more than a time quantum, the process runs for the full length of the time quantum and then it is preempted. The preempted process is then added to the tail of the queue again but with its CPU burst now a time quantum less than its previous CPU burst. This continues until the execution of the process is completed [1]. The RR algorithm is naturally preemptive [17].RR algorithm is one of the best scheduling algorithms that developed by many researchers [20-23].

2.3. Multiobjective optimization

The notion of weighing tradeoffs is common to problems in everyday life, science, and engineering. Buying a less expensive product might tradeoff product quality for the ability to buy more of something else. Adding an additional science instrument to a spacecraft trades off increased costs for increased science return. Hard optimization problems typically require many decisions on the input side and many objectives to optimize on the output side. The set of objectives forms a space where points in the space represent individual solutions. The goal of course is to find the best or optimal solutions to the optimization problem at hand. Pareto optimality defines how to determine the set of optimal solutions. A solution is Pareto-optimal if no other solution can improve one objective function without a simultaneous deterioration of at least one of the other objectives [24].

Def.1: (Multiobjective Optimization) A general MOP includes *a* set of *n* parameters (decision variables), *a* set of *k* objective functions, and *a* set of *m* constraints. Objective functions and constraints are functions of the decision variables.

The optimization goal is to

$$\begin{aligned} &\textit{Maximize} && y = f(x) = (f_1(x), f_2(x),..., f_k(x)) \\ &\textit{subject to} && e(x) = (e_1(x), e_2(x),..., e_m(x)) \le 0 \\ &\textit{where} && x = (x_1, x_2,..., x_n) \in X \\ & && y = (y_1, y_2,..., y_k) \in Y \end{aligned} \quad (1$$

and x is the decision vector, y is the objective vector, X is denoted as the decision space, and Y is called the objective space. The constraints $e(x) \le 0$ determine the set of feasible solutions.

Def.2: (Feasible Set) The feasible set X_f is defined as the set of decision vectors x that satisfy the constraints $e(x)$:

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$$X_f = \{x \in X \mid e(x) \leq 0\} \quad (2$$

The image of X_f , i.e., the feasible region in the objective space, is denoted as $Y_f = f(X_f) = \bigcup x \in X_f \{f(x)\}$.

Without loss of generality, a maximization problem is assumed here. For minimization or mixed maximization/minimization problems the definitions presented in this section are similar. Consider again the above example and assume that the two objectives performance (*f*1) and cheapness (*f*2), the inverse of cost, are to be maximized under size constraints (*e*1). Then an optimal design might be an architecture which achieves maximum performance at minimal cost and does not violate the size limitations. If such a solution exists, we actually only have to solve a single objective optimization problem (SOP). The optimal solution for either objective is also the optimum for the other objective. However, what makes MOPs difficult is the common situation when the individual optima corresponding to the distinct objective functions are sufficiently different. Then, the objectives are conflicting and cannot be optimized simultaneously. Instead, a satisfactory trade-off has to be found. In our example, performance and cheapness are generally competing: high-performance architectures substantially increase cost, while cheap architectures usually provide low performance. Depending on the market requirements, an intermediate solution (medium performance, medium cost) might be an appropriate trade-off. This discussion makes clear that a new notion of optimality is required for MOPs [25].

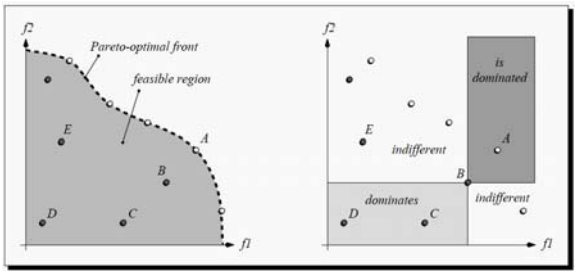

Figure 2: *Illustrative example of Pareto optimality in objective space (left) and the possible relations of solutions in objective space (right).*

In single-objective optimization, the feasible set is completely (totally) ordered according to the objective function *f:* for two solutions *a*; $b \in X_f$ either $f(a) \geq f(b) or f(b) \geq f(a)$.The goal is to find the solution (or solutions) that gives the maximum value of *f*. However, when several objectives are involved, the situation changes: X_f is, in general, not totally ordered, but partially ordered [26]. This is illustrated in Figure 2 on the left. The solution represented by point *B* is better than the solution represented by point *C*: it provides higher performance at lower cost. It would be even preferable if it would only improve one objective, as is the case for *C* and *D*: despite equal cost, *C* achieves better performance than *D*. In order to express this situation mathematically, the relations $=, \geq,$ and > are extended to objective vectors by analogy to the single-objective case.

Def. 3: (Pareto Dominance) For any two decision vectors *a* and *b*,

$a \succ b$(*a* dominates b) *iff* $f(a) > f(b)$

$a \succeq b$(*a weakly* dominates b) *iff* $f(a) \geq f(b)$

$a \approx b$(*a is indifferent to b*)

iff $f(a) \not\geq f(b) \cap f(b) \not\geq f(a)$

The definitions for a minimization problem $(\prec, \preceq, \approx)$ are analogical.

2.3.1. Fonseca and Fleming's Genetic Algorithm

Fonseca and Fleming proposed a Pareto-based ranking procedure (here the acronym FFGA is used) [26], where an individual's rank equals the number of solutions encoded in the population by which its corresponding decision vector is dominated. The fitness assignment procedure (For each individual $i \in p_t$ *do* $F(i) = f_i(m(i))$, which slightly differs from Goldberg's suggestion, consists of three steps:

(Fitness Assignment in FFGA)

Input: P_t (population)

σ_{share}(*niche radius*)

Output: F (fitness values)

Step 1: for each $i \in P_t$ calculate its rank: $r(i) = 1 + |\{j \mid j \in P_t \cap j \succ i\}|$.

Step 2: Sort population according to the ranking *r*. Assign each $i \in P_t$ a raw fitness $F'(i)$ by interpolating from the best (r (i) =1) to the worst individual $(r(i) \leq N)$; in this work linear ranking (Baker 1985) is used.

Step 3: Calculate fitness values F (i) by averaging and sharing the raw fitness values $F'(i)$ among

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

individuals $i \in P_t$ having identical ranks r (i) (fitness sharing in objective space).

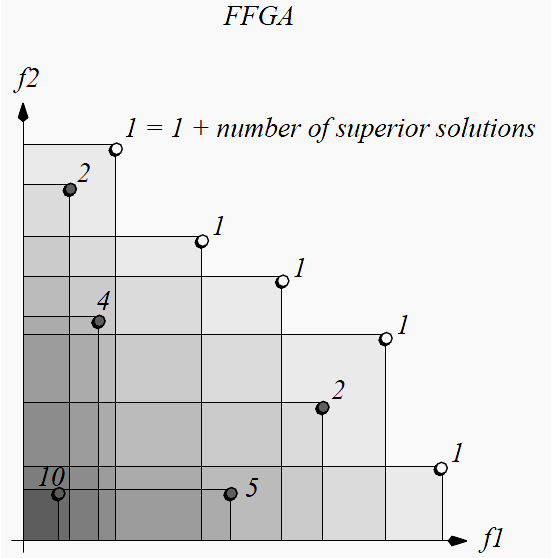

***Figure.3**: Fonseca and Fleming's Genetic Algorithm*

Note that the symbol |. | used in Step 1 denotes the number of elements in *Z* in conjunction with a (multi) set *Z*.

In Figure 3, a hypothetical population and the corresponding ranks of the individuals are shown. The individuals whose associated solutions are nondominated regarding *m (P)* have rank 1 while the worst individual was assigned rank 10. Based on the ranks, the mating pool is filled using stochastic universal sampling (Baker 1987).

The basic concept has been extended meanwhile by, e.g., adaptive fitness sharing and continuous introduction of random immigrants [27]-[28], which is, however, not regarded here.

2.4. Proposed Algorithm

In classical CPU scheduling algorithms, the only parameter used for process scheduling is the CPU burst time. In the proposed method for the process scheduling the three parameters of CPU burst time, I/O Service time, And priority of processes are used. In case only CPU burst time is taken into account for scheduling the algorithm would not notice the processes which need I/O and the system may not have a proper behavior or vice versa.

Any process which enters the system is surveyed from several points of view:

a) CPU burst time: Turnaround defines how long they need CPU to complete the operation. Prediction of this time is, however, difficult for the operating system. In fact the operating system makes a primary estimation and after running the process corrects this Estimation (The estimated value is updated). You may have noticed while trying to copy a CD the windows operating system firstly estimates a time value for the copying process but it does not take that long and this estimation is always longer than the real time. The more the copying process continues, the more accurate the estimation will be.

This value is in fact the next calculation burst. It is usually predicted as exponential mean of the lengths determined in previous bursts. Presuming that the length of Next Bursts is similar to the length of previous bursts it can be roughly calculated from the length of previous bursts. To do this the following formula is used:

$$T_{n+1} = \alpha t_n + (1-\alpha)T_n \quad (4$$

t_n = the length of n[th] real calculation burst

T_n = The predicted value for the n[th] calculation burst.

T_{n+1} = The predicted value for the next burst

α = The relative weight of recent and previous date of prediction.

If $\alpha = 1/2$ Recent and previous dates weigh equally.

If $\alpha = 1$ The recent date is not included in the calculation of the next burst time and only the previous date is included.

T_0 value can be defined as a constant or total mean of the system.

b) I/O service time: Any process might need to receive new data from the input unit or produce information results in the output unit while running. At this time the process no longer needs the CPU and if this process is the only process of the system the CPU will be idle. This situation is not considered appropriate by the system. The policy in designing the operating systems is to keep the CPU busy running a process and stop wasting its time.

It is easier to determine I/O service time than CPU burst time. Since the I/O unit is slower than the CPU it will therefore have a longer time. This time is calculated from the following formula by default:

$$P_n = S_n - T_N \quad (5)$$

S_n = the total time of running the process from the moment of arrival at the system to the moment it exits the system.

2.4.1. Pareto-optimal front

In MOO problems there are always several parameters for decision making and optimization

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

which sometimes conflict. In the proposed method for processes scheduling three parameters are used which sometimes Conflict. These processes are located in a three dimensional environment and have special positions according to their values. Those processes are selected from among processes which are not dominated by other processes [29] and these are the acceptable answers. Again, a process which is called the preferred answer is selected from among these processes according to the criteria and it is deleted from the set. Finally the priority of the process is also included so that the processes which are not dominated by any process are selected and placed in Pareto front.

2.4.2. Algorithm

The fitness processes and determination of Pareto front have been evaluated so far. The proposed algorithm will be described as follows.

(ACSFFGA)

1) Label 2: Input: p_n , pareto =null

2) Output:

$p_i \in p_n \quad and \quad p_i = \max(p_1, p_2, ..., p_n)$

3) For Each $p_i \in p_n$ Calculate

$rank : r(i) = 1 + \left|\{j | j \in p_n \ \& \ j > i\}\right|$

4) Sort r (i);

5) Select $r(i) = 1 \& insert \quad p_i \rightarrow pareto$

6) Label1: If CPU =0 then select

$\max(x \in p_i \ \& \ p_i \in pareto)$

7) If CPU =1 then select

$\max(y \in p_i \ \& \ p_i \in pareto)$

8) Run p_i ;

9) If p_i not completed run then

10) Remove $p_i \in pareto$

11) Update $p_i(x, y)$

12) Insert $p_i \in p_n$

13) Else

14) Remove $p_i \in pareto$

15) If Pareto! = null then

16) Next p_i

17) go to label1;

18) Else

19) For each $p_i \in p_n \mid p_i(priority) + 1$

20) Go to label 2

1) In the first step the program input is determined which is the p_n of the set of processes.

2) The program output which is the best process. It is the best process in Pareto front of the operating system and should control the CPU.

Tip: (Elitism which is put forward in MOO in this algorithm indicates in each application of the algorithm the elite process is found and deleted from the population.)

3) We score each process using FFGA algorithm (by the use of the dominate concept) and in fact the rank function determines the best processes which have the highest priority.

4) After determination of the fitness of each process, these processes are sorted (in ascending order).

5) The Pareto set is selected from among the processes with the lowest amount of rank (i) function (rank (i) =1).

6) If the CPU is free choose a process which has the shortest time needed for running.

7) If the CPU is busy running another process, choose a process which has the shortest time needed to communicate to the I/O unit so that I/O finishes its task while this process is running the CPU unit tasks.

8) In each case run the chosen process.

9) If running the process is not finished

10) Delete the chosen process from the Pareto set

11) Then reset the values for x, y (CPU burst time, I/O service time). In such cases the CPU burst time is less than before.

12) add the Deleted process from the Pareto front to the main set of p_n so that in the next generation the algorithm is applied to it again.

13, 14) However, if running the process is completed then the process is deleted from the Pareto front.

15, 16 and 17) if the Pareto front still has a process repeat the algorithm on the next process and go to step six.

18, 19) If the Pareto front has no process, increase the priority of all processes so that processes with lower priority would not face starvation. After a while when their aging is increased they will have higher priorities and will be included in the Pareto front. In fact the waiting time for processes is an auxiliary lever for selection.

20) Repeat the algorithm.

2.4.3. Specific conditions

We will face different conditions in the problem while running this algorithm which should be predicted in program. These conditions are described as follows:

2.4.3.1. Processes having requirements with the same angle coefficient

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

When the processes requirements coincide with the x=y axis, determination of the best process is very important for controlling system resources. According to a general rule, a process is appropriate which has shorter requirements. In classic cases this requirement is the only CPU burst time, however, in this algorithm both CPU and I/O unit times are taken into account.

A process is selected which is not dominated by any other process. However, in cases where processes' requirements have the same angle coefficient only one process is always selected since certainly a process is selected which has both smaller parameters.

2.4.3.2. Processes with the same priority in CPU burst time

We face a new state of processes now. This state happens when ready queuing processes all have the same requirements to CPU burst time. The General (Classic) algorithm selects the process which has a longer waiting time, but the proposed algorithm first puts all the processes in the Pareto front. Then it selects a process depending on the situation of the I/O whether it has a heavy traffic load or not (through adaptation). If the I/O unit has a heavy traffic load, a process with a short I/O time is selected; otherwise, a process with a high I/O unit demand will be selected.

* In fact the situation of I/O unit as well as the priority determines the selection method.

2.4.3.3. Processes with the same requirement for the I/O unit

This algorithm may apparently act the same as the classic algorithm here, while in fact it is not so. In classic algorithms a process with fewer requirements for CPU is selected in this case without regarding the correct system computational volume. If the system has a small computational load then it can easily perform a process with high computational volume in a short period of time. Bear in mind that the CPU performance rate is several million instructions per second, and its performance structure is different from that of I/O unit. When a system is busy with a high computational load, however, the best choice is to select a process with a short CPU burst time.

2.4.3.4. Processes with symmetric requirements

Presume four processes with symmetric requirements are located at the four corners of a square. If we want to act according to the classic pattern, the process at the top right corner is immediately selected while the process at the top left corner has the same need to CPU. In the algorithm with MOO technology both parameters are taken into account and the processes which are subordinate to rank (p) =1 is placed in the Pareto front. The preferable process is selected depending on the situation of CPU, and I/O unit.

2.4.3.5. Using clustery in determination of Pareto front

We sometimes face different behaviors of processes. One of these behaviors is the appearance of two different types of behavior; i.e. some processes have the same CPU burst time, and some have the same I/O unit service time. Classical algorithms select a process with the shortest CPU burst time which will have a very weak performance here. The proposed algorithm in this case selects a process considering the two resources of CPU, and I/O unit, as well as priority. It should be noted that in such cases all processes are included in Pareto front, and all processes have the chance of being selected.

After selection of the front and determination of the system conditions a dominant process will be selected. In the world of genetics, elitism we always hope that elite people survive during genetic mutations and don't perish, but in this system dominant processes leave the system soon. The system aims at deleting these processes and run them as soon as possible.

3. DATA

In this study 50 processes with the characteristics presented in Table (1) have been used. Each process has three parameters defined as follows [30]:

Initial burst time: this is the amount of CPU time the process will require. In real life this is not really known, but can be predicted with some degree of accuracy.

Delay (I/O): the time separating the arrival of processes. The amount of time after the (n-1)th process arrives that this process arrives.

Priority: For prioritized algorithms this is the relative weight of this process. The range is from 0 - 9 where 9 is the lowest priority and 0 is the highest.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table.1 Data set

#	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}
Burst	7	24	37	13	3	56	25	20	15	17	29	6	14	26	48	8	28
I/O	70	70	95	25	45	55	95	10	45	60	15	70	20	40	25	15	20
Priority	5	4	8	3	0	6	1	1	2	5	7	0	1	0	9	2	2
#	P_{18}	P_{19}	P_{20}	P_{21}	P_{22}	P_{23}	P_{24}	P_{25}	P_{26}	P_{27}	P_{28}	P_{29}	P_{30}	P_{31}	P_{32}	P_{33}	P_{34}
Burst	59	33	96	28	65	37	99	22	13	23	56	30	7	99	29	32	4
I/O	15	50	85	75	50	0	5	40	70	55	55	25	10	10	45	35	20
Priority	0	9	0	0	4	2	8	7	3	8	4	6	1	77	6	2	2
#	P_{35}	P_{36}	P_{37}	P_{38}	P_{39}	P_{40}	P_{41}	P_{42}	P_{43}	P_{44}	P_{45}	P_{46}	P_{47}	P_{48}	P_{49}	P_{50}	
Burst	21	19	44	36	9	39	21	41	15	36	38	12	4	37	34	17	
I/O	90	10	70	65	15	30	5	15	15	25	95	65	10	5	55	10	
Priority	2	3	2	1	1	8	0	6	7	1	0	0	9	5	7	3	

Table.2 First-Come, First-Served (FCFS)

#	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}
Start	70	140	235	272	305	360	455	480	510	570	587	655	675	715	741	789	797
Finish	76	163	271	284	307	415	479	499	524	586	615	660	688	740	788	796	824
Wait	0	0	0	12	0	0	0	15	0	0	2	0	0	0	1	34	22
Response	0	0	0	12	0	0	0	15	0	0	2	0	0	0	1	34	22
Turnaround	7	24	37	25	3	56	25	35	15	17	31	6	14	26	49	42	50
#	P_{18}	P_{19}	P_{20}	P_{21}	P_{22}	P_{23}	P_{24}	P_{25}	P_{26}	P_{27}	P_{28}	P_{29}	P_{30}	P_{31}	P_{32}	P_{33}	P_{34}
Start	825	884	925	1021	1050	1115	1152	1251	1273	1286	1309	1365	1395	1402	1501	1530	1562
Finish	883	916	1020	1048	1114	1151	1250	1272	1285	1308	1364	1394	1401	1500	1529	1561	1565
Wait	35	44	0	21	0	65	97	165	108	66	34	65	85	82	136	130	142
Response	35	44	0	21	0	65	97	165	108	66	34	65	85	82	136	130	142
Turnaround	94	77	96	49	65	102	196	178	121	89	90	95	92	181	165	162	146
#	P_{35}	P_{36}	P_{37}	P_{38}	P_{39}	P_{40}	P_{41}	P_{42}	P_{43}	P_{44}	P_{45}	P_{46}	P_{47}	P_{48}	P_{49}	P_{50}	
Start	1566	1587	1606	1655	1691	1700	1739	1760	1801	1816	1855	1920	1932	1936	1990	2024	
Finish	1586	1605	1649	1690	1699	1738	1759	1800	1815	1851	1892	1931	1935	1372	2023	2040	
Wait	56	67	16	0	21	0	34	40	66	56	0	0	2	1	0	24	
Response	56	67	16	0	21	0	34	40	66	56	0	0	2	1	0	24	
Turnaround	77	86	60	36	30	39	55	81	81	92	38	12	6	38	34	41	

Table.3 Priority Weighted (Non-Preemptive)

#	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}
Start	70	140	235	272	305	360	455	480	510	570	587	655	675	715	741	789	856
Finish	76	163	271	284	307	415	479	499	524	586	615	660	688	740	788	796	883
Wait	0	0	0	12	0	0	0	15	0	0	2	0	0	0	1	34	81
Response	0	0	0	12	0	0	0	15	0	0	2	0	0	0	1	34	81
Turnaround	7	24	37	25	3	56	25	35	15	17	31	6	14	26	49	42	109
#	P_{18}	P_{19}	P_{20}	P_{21}	P_{22}	P_{23}	P_{24}	P_{25}	P_{26}	P_{27}	P_{28}	P_{29}	P_{30}	P_{31}	P_{32}	P_{33}	P_{34}
Start	797	884	925	1021	1087	1050	1187	1152	1174	1583	1286	1349	1342	1444	1379	1408	1440
Finish	855	916	1020	1048	1151	1086	1285	1173	1186	1605	1341	1378	1348	1542	1407	1439	1443
Wait	7	44	0	21	37	0	132	57	9	363	11	49	32	124	14	8	20
Response	7	44	0	21	37	0	132	57	9	363	11	49	32	124	14	8	20
Turnaround	66	77	96	49	102	37	231	79	22	386	67	79	39	223	43	40	24
#	P_{35}	P_{36}	P_{37}	P_{38}	P_{39}	P_{40}	P_{41}	P_{42}	P_{43}	P_{44}	P_{45}	P_{46}	P_{47}	P_{48}	P_{49}	P_{50}	
Start	1543	1564	1606	1655	1691	1700	1739	1796	1837	1760	1855	1920	1932	1936	1990	2024	
Finish	1563	1582	1649	1690	1699	1738	1759	1836	1851	1795	1892	1931	1935	1972	2023	2040	
Wait	33	44	16	0	21	0	34	76	102	0	0	0	2	1	0	24	
Response	33	44	16	0	21	0	34	76	102	0	0	0	2	1	0	24	
Turnaround	54	63	60	36	30	39	55	117	117	36	38	12	6	38	34	41	

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4. EXPERIMENTAL RESULTS

The proposed method has been tested using the data from Table (1). Start, finish, wait, response and turnaround parameters have been reviewed. The proposed method was compared with 7 classical scheduling algorithms. These algorithms are as follows:

4.1. First-Come, First-Served (FCFS)

This algorithm selects the simplest method for CPU scheduling. A process which enters the queue earlier will control the CPU earlier. This method is non-preemptive. This method usually has a long waiting time on average. Table (2) shows the results of this algorithm.

In the FCFS algorithm the waiting time and response time are the same. As it could be seen in Table (2), the waiting time is usually long and fairness is not observed among the processes. For example, P_{34} process which only needs the CPU for 4 seconds that should wait for 142 seconds.

4.2. Priority Weighted (PRI) (Preemptive, Non-preemptive)

The priority scheduling algorithm selects its next job based on the importance of the process. The job that has the highest priority (0 high: 9 low) is run first. If preemption is enabled the new jobs with a higher priority will interrupt the currently executing job. Without preemption the highest priority job is chosen after the active process finishes execution.

The PRI method makes decisions regarding the priority of the processes. Determination of the processes priority rests on the operating system. When a process enters the ready processes queue, its priority is determined and will not change to the end. This could be a weak point for this method.

One of the problems of preemptive CPU scheduling algorithms is the period of time a scheduler should switch between the processes. This causes an overhead to be applied to the CPU and System performance is decreased. After reviewing the results of Tables (3) and (4) it could clearly be seen that in PRI (non-preemptive) method the average waiting time and the average Turnaround time are less than those of PRI (preemptive) method. The non-preemption of the CPU has also some advantages including the ability to reduce the response time for the processes which can be viewed in the results of tables (3) and (4). The PRI method can generally be a fair and good method.

4.3. Round Robin (RR) (Prioritized, Equal time)

The Round Robin scheduling algorithm allocates a time slice to each running process. This is called the quantum and it represents the number of CPU cycles a process gets before the scheduler searches for a new job to run. Jobs receive their quantum of CPU time in FCFS order. With priority scheduling enabled the quantum is multiplied by the magnitude of a processes priority. Thus more important jobs get more CPU time.

The RR method has some advantages including the average short response time of the processes since this method switches between the processes using the time slices. But the RR method applies an extra overhead to the CPU due to these multiple switches and causes an increase in the average Turnaround time. Whit attention to table(5) and table(6), the performance of RR algorithm with identical time slices is better than the RR method with Prioritized.

4.4. Shortest Job First (SJF) (Preemptive, Non-preemptive)

The SJF algorithm chooses the shortest job. Without preemption jobs run to completion before a new job is selected. If preemption is enabled then the instant a job arrives that is shorter than the one being run the CPU switches to the new shorter job. The processing can be interrupted to run newly arrived shorter jobs.

This method only pay attention to the CPU burst time and this filed is checked. This algorithm is presented in two states of non-preemptive and preemptive. Regarding the Tables (7) and (8) the performance of preemptive method is preferable. The SJF method has shown a better performance compared to previous methods, and it could be concluded that the processes which have a shorter CPU burst time would run earlier.

Journal of Theoretical and Applied Information Technology
15th March 2012. Vol. 37 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table.4 Priority Weighted (Preemptive)

#	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}
Start	70	140	235	260	305	360	455	480	510	570	587	655	675	715	741	755	775
Finish	76	163	284	272	307	415	479	499	524	586	615	660	688	740	883	762	861
Wait	0	0	13	0	0	0	0	15	0	0	2	0	0	0	96	0	59
Response	0	0	0	0	0	0	0	15	0	0	2	0	0	0	1	0	0
Turnaround	7	24	50	13	3	56	25	35	15	17	31	6	14	26	144	8	87
#	P_{18}	P_{19}	P_{20}	P_{21}	P_{22}	P_{23}	P_{24}	P_{25}	P_{26}	P_{27}	P_{28}	P_{29}	P_{30}	P_{31}	P_{32}	P_{33}	P_{34}
Start	790	884	925	1021	1087	1050	1187	1152	1165	1583	1275	1338	1310	1397	1368	1400	1432
Finish	848	916	1020	1048	1151	1086	1582	1186	1177	1649	1337	1367	1316	1571	1396	1431	1435
Wait	0	44	0	21	37	0	429	70	0	407	7	38	0	153	3	0	12
Response	0	44	0	21	37	0	132	57	0	363	0	38	0	77	3	0	12
Turnaround	59	77	96	49	102	37	528	92	13	430	63	68	7	252	32	32	16
#	P_{35}	P_{36}	P_{37}	P_{38}	P_{39}	P_{40}	P_{41}	P_{42}	P_{43}	P_{44}	P_{45}	P_{46}	P_{47}	P_{48}	P_{49}	P_{50}	
Start	1510	1531	1590	1655	1691	1700	1705	1726	1803	1760	1855	1920	1932	1935	1990	2000	
Finish	1530	1549	1633	1690	1699	1851	1725	1802	1817	1795	1892	1931	1972	1971	2040	2016	
Wait	0	11	0	0	21	113	0	42	68	0	0	0	39	0	17	0	
Response	0	11	0	0	21	0	0	6	68	0	0	0	2	0	0	0	
Turnaround	21	30	44	36	30	152	21	83	83	36	38	12	43	37	51	17	

Table .5 Round Robin (Equal time)

#	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}
Start	70	140	235	265	305	360	455	465	510	570	587	655	675	715	741	761	779
Finish	76	163	281	284	307	415	499	494	524	586	615	660	688	740	826	768	834
Wait	0	0	10	12	0	0	20	10	0	0	2	0	0	0	39	6	32
Response	0	0	0	5	0	0	0	0	0	0	2	0	0	0	1	6	4
Turnaround	7	24	47	25	3	56	45	30	15	17	31	6	14	26	87	14	60
#	P_{18}	P_{19}	P_{20}	P_{21}	P_{22}	P_{23}	P_{24}	P_{25}	P_{26}	P_{27}	P_{28}	P_{29}	P_{30}	P_{31}	P_{32}	P_{33}	P_{34}
Start	809	845	925	1005	1050	1060	1070	1110	1209	1257	1287	1320	1330	1366	1376	1426	1436
Finish	916	907	1040	1048	1246	1176	1345	1186	1241	1309	1474	1405	1336	1615	1468	1546	1439
Wait	68	35	20	21	132	90	192	72	64	67	144	76	20	197	75	115	16
Response	19	5	0	5	0	10	15	15	44	37	12	20	20	46	11	26	16
Turnaround	127	68	116	49	197	127	291	94	77	90	200	106	27	296	104	147	20
#	P_{35}	P_{36}	P_{37}	P_{38}	P_{39}	P_{40}	P_{41}	P_{42}	P_{43}	P_{44}	P_{45}	P_{46}	P_{47}	P_{48}	P_{49}	P_{50}	
Start	1515	1525	1597	1655	1675	1700	1710	1720	1760	1770	1855	1920	1930	1936	1990	2000	
Finish	1576	1565	1649	1699	1683	1814	1790	1835	1805	1851	1892	1935	1933	1972	2040	2026	
Wait	46	27	16	9	5	76	65	75	56	56	0	4	0	1	17	10	
Response	5	5	7	0	5	0	5	0	25	10	0	0	0	1	0	0	
Turnaround	67	46	60	45	14	115	86	116	71	92	38	16	4	38	51	27	

Table.6 Round Robin (Prioritized)

#	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}
Start	70	140	235	272	305	360	455	465	510	570	587	655	675	715	741	789	797
Finish	76	163	271	284	307	415	499	494	524	586	615	660	688	740	788	796	883
Wait	0	0	0	12	0	0	20	10	0	0	2	0	0	0	1	34	81
Response	0	0	0	12	0	0	0	0	0	0	2	0	0	0	1	34	22
Turnaround	7	24	37	25	3	56	45	30	15	17	31	6	14	26	49	42	109
#	P_{18}	P_{19}	P_{20}	P_{21}	P_{22}	P_{23}	P_{24}	P_{25}	P_{26}	P_{27}	P_{28}	P_{29}	P_{30}	P_{31}	P_{32}	P_{33}	P_{34}
Start	817	884	925	1021	1050	1090	1110	1190	1273	1286	1309	1349	1379	1386	1456	1485	1505
Finish	875	916	1020	1048	1236	1253	1272	1211	1285	1308	1524	1378	1385	1553	1484	1565	1508
Wait	27	44	0	21	122	167	119	95	108	66	194	49	69	135	91	134	85
Response	27	44	0	21	0	40	55	95	108	66	34	49	69	66	91	85	85
Turnaround	86	77	96	49	187	204	218	117	121	89	250	79	76	234	120	166	89
#	P_{35}	P_{36}	P_{37}	P_{38}	P_{39}	P_{40}	P_{41}	P_{42}	P_{43}	P_{44}	P_{45}	P_{46}	P_{47}	P_{48}	P_{49}	P_{50}	
Start	1566	1586	1606	1655	1675	1700	1739	1760	1801	1816	1855	1920	1932	1936	1990	2024	
Finish	1605	1604	1649	1699	1683	1738	1759	1800	1815	1851	1892	1931	1935	1972	2023	2040	
Wait	75	66	16	9	5	0	34	40	66	56	0	0	2	1	0	24	
Response	56	66	16	0	5	0	34	40	66	56	0	0	2	1	0	24	
Turnaround	96	85	60	45	14	39	55	81	81	92	38	12	6	38	34	41	

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

This method only pay attention to the CPU burst time and this filed is checked. This algorithm is presented in two states of non-preemptive and preemptive. Regarding the Tables (7) and (8) the performance of preemptive method is preferable. The SJF method has shown a better performance compared to previous methods, and it could be concluded that the processes which have a shorter CPU burst time would run earlier.

4.5. Proposed algorithm

In the proposed method the three parallel parameters of (CPU, and I/O service time, and priority) are taken into account. And using the FFGA multiobjective optimization method the processes are selected after determination of fitness.

While running the proposed algorithm we encounter certain conditions which show its performance well. Some of these conditions are described as follows:

a: P_3 process arrives at the system at the moment 235, and since there is no process it begins to run. After 25 seconds the process P_4 arrives at the system and there is a competition between the two processes. The proposed algorithm reviews the two processes. Since P_4 has the (burst=13) and the priority of 3, and P_3 has (burst=12 (remaining) and priority of 8) the proposed algorithm selects the P_4 as the winner and retrieves the CPU from P_3 and passes it to P_4 to get control of the CPU. P_4 will have the waiting and response time of zero. After running of P_4 is completed, the CPU is passed to P_3 again so that its execution is completed.

b: P_7 arrives at the system at the moment 453 and controls CPU, but at the moment 465 P_8 process arrives at the system. Both processes have the same priority. The remaining execution of P_7 is only 15 seconds, while for P_8 burst=20, therefore, the algorithm continues to run P_7.

c: P_{10} process arrives at the system at the moment 510 and starts running, but at the moment 585 the P_{11} process arrives at the system. This process has the priority of 7 and burst=21 while P_{10} process has the priority of 5 and burst=2(remaining). The algorithm keeps the P_{11} in waiting state so that running of P_{10} completes.

d: P_{14} process starts running at the moment 715 to 740 when P_{15} process arrives at the system. P_{15} has the priority of 9 and burst=48 while P_{14} has the priority of zero and burst=1. The scheduling algorithm continues running P_{14} process until it completes.

e: P_{15} process starts running at the moment 741, but at the moment 756 the P_{16} process arrives at the system. This process has the priority of 2 and burst=2, while P_{15} has the priority of 9 and burst=33(remaining). The proposed algorithm stops running P_{15} and passes the CPU to P_{16}. Running of P_{16} completes at the moment 762. The system resumes running P_{15}, but at the moment 775 P_{17} process arrives at the system. This process has the priority of 2 and burst=28, while P_{15} has the priority of 9 and burst=20 (remaining). The proposed algorithm stops running P_{15} and passes the CPU to P_{17}. At the moment 790 the P_{18} process arrives at the system. Now the algorithm can select three processes P_{17}, P_{18} and P_{15} where regarding the parameters of the processes the P_{17} process continues running to be completed. Running of P_{17} completes at the moment 802. Now there are two processes of P_{15}, and P_{18} available in the system. Regarding the passage of time, the proposed algorithm increases the priority of the P_{15} process and P_{15} starts running until the moment 822.

f: P_{20} process arrives at the system at the moment 925 and starts running, but at the moment 1000 the P_{21} process arrives at the system. The P_{21} process has the priority of zero and its burst=28, and the P_{20} process has the priority of zero and its burst=21. The proposed algorithm continues running P_{20}.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table.7 Shortest job first

#	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}
Start	70	140	235	272	305	360	455	480	510	570	587	655	675	715	741	789	797
Finish	76	163	271	284	307	415	479	499	524	586	615	660	688	740	788	796	824
Wait	0	0	0	12	0	0	0	15	0	0	2	0	0	0	1	34	22
Response	0	0	0	12	0	0	0	15	0	0	2	0	0	0	1	34	22
Turnaround	7	24	37	25	3	56	25	35	15	17	31	6	14	26	49	42	50
#	P_{18}	P_{19}	P_{20}	P_{21}	P_{22}	P_{23}	P_{24}	P_{25}	P_{26}	P_{27}	P_{28}	P_{29}	P_{30}	P_{31}	P_{32}	P_{33}	P_{34}
Start	825	884	925	1021	1087	1050	1187	1152	1174	1286	1346	1309	1339	1467	1402	1435	1431
Finish	883	916	1020	1048	1151	1086	1285	1173	1186	1308	1401	1338	1345	1565	1430	1466	1434
Wait	35	44	0	21	37	0	132	57	9	66	71	9	29	147	37	35	11
Response	35	44	0	21	37	0	132	57	9	66	71	9	29	147	37	35	11
Turnaround	94	77	96	49	102	37	231	79	22	89	127	39	36	246	66	67	15
#	P_{35}	P_{36}	P_{37}	P_{38}	P_{39}	P_{40}	P_{41}	P_{42}	P_{43}	P_{44}	P_{45}	P_{46}	P_{47}	P_{48}	P_{49}	P_{50}	
Start	1585	1566	1606	1655	1691	1700	1754	1811	1739	1775	1855	1920	1932	1936	1990	2024	
Finish	1605	1584	1649	1690	1699	1738	1774	1851	1753	1810	1892	1931	1935	1972	2023	2040	
Wait	75	46	16	0	21	0	49	91	4	15	0	0	2	1	0	24	
Response	75	46	16	0	21	0	49	91	4	15	0	0	2	1	0	24	
Turnaround	96	65	60	36	30	39	70	132	119	51	38	12	6	38	34	41	

Table.8 Shortest job first (preemptive)

#	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}
Start	70	140	235	272	305	360	455	480	510	570	587	655	675	715	741	755	797
Finish	76	163	271	284	307	415	479	499	524	586	615	660	688	740	796	762	824
Wait	0	0	0	12	0	0	0	15	0	0	2	0	0	0	9	0	22
Response	0	0	0	12	0	0	0	15	0	0	2	0	0	0	1	0	22
Turnaround	7	24	37	25	3	56	25	35	15	17	31	6	14	26	57	8	50
#	P_{18}	P_{19}	P_{20}	P_{21}	P_{22}	P_{23}	P_{24}	P_{25}	P_{26}	P_{27}	P_{28}	P_{29}	P_{30}	P_{31}	P_{32}	P_{33}	P_{34}
Start	825	840	925	1021	1087	1050	1187	1095	1174	1220	1346	1309	1310	1467	1365	1435	1420
Finish	916	872	1020	1048	1173	1086	1308	1116	1186	1242	1434	1345	1316	1605	1393	1466	1423
Wait	68	0	0	21	59	0	155	0	9	0	104	16	0	187	0	35	0
Response	35	0	0	21	37	0	132	0	9	0	71	9	0	147	0	35	0
Turnaround	127	33	96	49	124	37	254	22	22	23	160	46	7	286	29	67	4
#	P_{35}	P_{36}	P_{37}	P_{38}	P_{39}	P_{40}	P_{41}	P_{42}	P_{43}	P_{44}	P_{45}	P_{46}	P_{47}	P_{48}	P_{49}	P_{50}	
Start	1510	1531	1606	1655	1670	1700	1705	1811	1735	1775	1855	1920	1932	1936	1990	2000	
Finish	1530	1549	1649	1699	1678	1774	1725	1851	1749	1810	1892	1931	1935	1972	2040	2016	
Wait	0	11	16	9	0	36	0	91	0	15	0	0	2	1	17	0	
Response	0	11	16	0	0	0	0	91	0	15	0	0	2	1	0	0	
Turnaround	21	30	60	45	9	75	21	132	15	51	38	12	6	38	51	17	

Table.9 proposed method results

#	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}
Start	70	140	235	260	305	360	455	480	510	570	587	655	675	715	741	755	775
Finish	76	163	285	273	307	415	479	499	524	586	615	660	688	740	822	762	802
Wait	0	0	0	0	0	0	0	15	0	0	2	0	0	0	21	0	0
Response	0	0	0	0	0	0	0	15	0	0	2	0	0	0	1	0	0
Turnaround	7	24	50	13	3	56	25	35	15	17	31	6	14	26	69	8	28
#	P_{18}	P_{19}	P_{20}	P_{21}	P_{22}	P_{23}	P_{24}	P_{25}	P_{26}	P_{27}	P_{28}	P_{29}	P_{30}	P_{31}	P_{32}	P_{33}	P_{34}
Start	822	840	925	1021	1087	1050	1187	1095	1174	1220	1346	1309	1310	1367	1365	1435	1420
Finish	913	872	1020	1048	1152	1086	1308	1116	1186	1242	1434	1355	1316	1476	1448	1466	1423
Wait	32	9	0	21	36	0	97	0	9	0	0	38	0	0	54	0	0
Response	32	9	0	21	36	0	97	0	9	0	0	31	0	0	54	0	0
Turnaround	91	42	96	49	101	37	196	22	22	23	56	46	7	99	83	32	4
#	P_{35}	P_{36}	P_{37}	P_{38}	P_{39}	P_{40}	P_{41}	P_{42}	P_{43}	P_{44}	P_{45}	P_{46}	P_{47}	P_{48}	P_{49}	P_{50}	
Start	1510	1531	1606	1655	1670	1700	1705	1811	1735	1775	1855	1920	1932	1936	1990	2000	
Finish	1530	1549	1649	1691	1678	1769	1725	1851	1749	1811	1892	1931	1935	1972	2040	2016	
Wait	0	11	0	9	0	30	0	60	0	0	0	0	2	1	17	0	
Response	0	11	0	0	0	0	0	60	0	0	0	0	2	1	0	0	
Turnaround	21	30	44	36	9	69	21	102	15	36	38	12	6	38	51	17	

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table10 comparing the performances of 7 scheduling algorithms with that of the suggested algorithm

	FCFS			Priority (non-preemptive)			Priority (preemptive)		
#	Wait	Response	Turnaround	Wait	Response	Turnaround	Wait	Response	Turnaround
Min	0	0	3	0	0	3	0	0	3
Mean	34.7	34.7	65.32	28.52	28.52	59.14	34.34	18.2	64.96
Max	156	156	196	363	363	386	429	363	528
Std.DEV	43.67	43.67	49.75	58.02	58.02	65.74	85.6	55.81	96.83
	Round Robin (equal time)			Round Robin (prioritized)			Shortest Job First		
#	Wait	Response	Turnaround	Wait	Response	Turnaround	Wait	Response	Turnaround
Min	0	0	3	0	0	3	0	0	3
Mean	39.96	7.64	70.58	41.6	27.44	72.22	23.4	23.4	54.02
Max	197	46	296	194	108	250	147	147	246
Std.DEV	48.8	11.44	65.81	50.42	32.24	61.46	33.49	33.49	48.95
#	Shortest Job First (preemptive)			ACSFFGA					
	Wait	Response	Turnaround	Wait	Response	Turnaround			
Min	0	0	3	0	0	3			
Mean	18.24	13.68	48.86	9.28	7.62	39.76			
Max	187	147	286	97	97	196			
Std.DEV	39.06	31.59	57.55	40.21	34.57	51.73			

G: The P_{22} process arrives at the system at the moment 1050 and immediately the P_{23} process also arrives. The P_{22} process has the priority of 4 and its burst=65, and the P_{23} process has the priority of 2 and its burst=37. The proposed algorithm runs the P_{23} process, but at the moment 1055 the P_{24} process also arrives at the system. Regarding the characteristics of the P_{24} process, the proposed algorithm continues running P_{23} and completes it. Now, it runs the P_{24} process from among the two remaining processes (P_{22} , P_{22}).

According to the Table 9, the proposed algorithm has the best results. The average waiting time for 50 processes is 9.28, the average response time is 7.62, and the average Turnaround time is 39.76. From the 7 scheduling algorithms, the best method is SJF (preemptive) method. The average waiting time for 50 processes is 18.24, the average response time is 13.68, and the average Turnaround time is 48.86. The SJF and Priority (non-preemptive**)** methods come in second and third, respectively.

5. CONCLUSION

One of the main parts of any operating system is the scheduler. The schedulers try to improve the performance of the system by allocation of the resources to processes. Many CPU scheduling algorithms have been presented having advantages and disadvantages each. In this paper a new CPU scheduling algorithm is presented. This algorithm schedules the running of processes according to the three parameters of CPU burst time, I/O service time, and priority of processes). To this end, the FFGA multiobjective optimization algorithm is used. The proposed algorithm selects and runs the desired processes through adaptation. In this algorithm, the priority of processes increases with time, and no process encounters starvation. A comparison of the proposed algorithm with the 7 other methods showed that the average waiting time and average response time are decreased. The average turnaround time is also improved. One of the most important positive points of this method is that there is fairness among processes. Using other methods of artificial intelligence such as Fuzzy Logic, Neural Network, and Swarm Intelligence will be covered by the authors in the future. We hope to improve the performance of the operating systems and use the maximum potential of processors by using more advanced scheduling algorithms.

REFRENCES:

[1] Silberschatz ,Galvin and Gagne, Operating systems concepts, 8th edition, Wiley, 2009.
[2] Lingyun Yang, Jennifer M. Schopf and Ian Foster, “Conservative Scheduling: Using

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

predictive variance to improve scheduling decisions in Dynamic Environments", Super Computing 2003, November 15-21, Phoenix, AZ, USA.

[3] Weiming Tong, Jing Zhao, "Quantum Varying Deficit Round Robin Scheduling over Priority Queues", international Conference on Computational Intelligence and Security. pp 252- 256, China, 2007.

[4] Abbas Noon1, Ali Kalakech2, Seifedine Kadry1, "A New Round Robin Based Scheduling Algorithm for Operating Systems: Dynamic Quantum Using the Mean Average", IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 3, No. 1, May 2011.

[5] SAROJ HIRANWAL and K. C. ROY, "ADAPTIVE ROUND ROBIN SCHEDULING USING SHORTEST BURST APPROACH BASED ON SMART TIME SLICE", International Journal of Computer Science and Communication Vol. 2, No. 2, July-December 2011, pp. 319-323.

[6] Ajit Singh, Priyanka Goyal, Sahil Batra," An Optimized Round Robin Scheduling Algorithm for CPU Scheduling", (IJCSE) International Journal on Computer Science and Engineering Vol. 02, No. 07, 2010, 2383-2385.

[7] Shatha J. Kadhim1 and Kasim M. Al-Aubidy," Design and Evaluation of a Fuzzy-Based CPU Scheduling Algorithm", BAIP 2010, CCIS 70, pp. 45 – 52, © Springer-Verlag Berlin Heidelberg 2010.

[8] Ozelkan, E.C., Duckestine, L.," Optimal Fuzzy Counterparts of Scheduling Rules", .European Journal of operational Research (113), 593–609 (1999).

[9] McCahon, C.S., Lee, E.S, "Job Sequencing with Fuzzy Processing Times", Computers & Mathematics with Applications (19), 31–41 (1990).

[10] Hapke, M., Slowinski, R., "Fuzzy Priority Heuristics for Project Scheduling", Fuzzy Sets and Systems (83), 291–299 (1996).

[11] A., Bashir, M.N. Doja and R., Biswas, "Conceptual Improvement of Priority Based CPU Scheduling Algorithm Using Fuzzy Logic," International Journal of Fuzzy Systems and Rough Systems (IJFSRS) (Vol. 1, No. 1, Ja-nu.-June 2008)

[12]George Kousiouris Tommaso Cucinotta, Theodora Varvarigou**,"** The effects of scheduling workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks **",** Journal of Systems and Software,Volume 84, Issue 8, August 2011, Pages 1270-1291.

[13] Carole Fayad and Sanja Petrovic," A Genetic Algorithm for the Real-World Fuzzy Job Shop Scheduling ", Innovations in Applied Artificial Intelligence , Springer, Pages: 524–533, 2005.

[14] Sanja Petrovic and Carole Fayad, "A Genetic Algorithm for Job Shop scheduling with Load Balancing ",*AI 2005, Lecture Notes in Artificial Intelligence*, 3809, Springer, 2005, pp. 339-348.

[15] Qun Niu, Bin Jiao, Xingsheng Gu,m ," Particle swarm optimization combined with genetic operators for job shop scheduling problem with fuzzy processing time", Applied Mathematics and Computation ,Volume 205, Issue 1, 1 November 2008, Pages 148-158.

[16] Surekha P, Dr.S.Sumathi, "PSO and ACO based approach for solving combinatorial Fuzzy Job Shop Scheduling Problem", Int. J. Comp. Tech. Appl., Vol 2 (1), 112-120.

[17] 1E.O. Oyetunji and 2A. E. Oluleye, "Performance Assessment of Some CPU Scheduling Algorithms", Research Journal of Information Technology 1(1): 22-26, 2009.

[18] Daniel Chillet, Antoine Eiche, Sébastien Pillement, Olivier Sentieys," Real-time scheduling on heterogeneous system-on-chip architectures using an optimised artificial neural network", Journal of Systems Architecture, Volume 57, Issue 4, April 2011, Pages 340-353.

[19] Coello, C. "A comprehensive survey of evolutionary-based multiobjective optimization techniques". *Knowledge and Information Systems 1* (3), 269–308, 1999.

[20] Shih-Chiang Tsao, Ying-Dar Lin , "Pre-order Deficit Round Robin : a new scheduling algorithm for packet-switched networks" , Computer Networks, Volume 35, Issues 2-3, February 2001, Pages 287-305.

[21] Alam, B.; Doja, M.N.; Biswas, R.; "Finding Time Quantum of Round Robin CPU Scheduling Algorithm Using Fuzzy Logic", International Conference on Computer and Electrical Engineering (ICCEE), Phuket , 795 – 798 , 2008 .

[22] Debashree Nayak, R Mohanty and H.S.Behera. Article: A New Proposed Dynamic Quantum with Re-Adjusted Round Robin Scheduling Algorithm and Its Performance Analysis. International Journal of Computer Applications 5(5):10–15, August 2010. Published By Foundation of Computer Science.

[23] Tech. Univ. of Denmark, Lygnby, Denmark , "A Novel Round-Robin Based Multicast Scheduling Algorithm for 100 Gigabit Ethernet

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Switches " , INFOCOM IEEE Conference on Computer Communications Workshops , 2010 , San Diego, CA,2010.

[24] Jason D. Lohn, William F. Kraus, Gary L. Haith," Comparing a Co evolutionary Genetic Algorithm for Multiobjective Optimization", proceeding of the IEEE Congress on Evolutionary Computation, may 2002, pp.1157-1162.

[25] Eckart Zitzler," Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications", Swiss Federal Institute of Technology Zurich for the degree of Doctor of Technical Sciences, Diss. ETH No. 13398, 1999.

[26] Fonseca, C. M. and P. J. Fleming. "Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization". In S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo, California, pp. 416–423. Morgan Kaufmann, 1993.

[27] Fonseca, C. M. and P. J. Fleming." Multiobjective genetic algorithms made easy: Selection, sharing and mating restrictions". In *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA 95)*, London, UK, pp. 45–52. The Institution of Electrical Engineers, 1995.

[28] Fonseca, C. M. and P. J. Fleming. Multiobjective optimization and multiple constraints handling with evolutionary algorithms—part i: A unified formulation. *IEEE Transactions on Systems, Man, and Cybernetics 28*(1), 26–37, 1998.

[29] N.srinivas, Kalyanmoy Deb, "Multiobjective Optimization using nondominated sorting in genetic algorithms" .journal of evolutionary computation, vol .2, no.3, pages 221-248.

[30] Jim weller ,CPU Scheduler Application , http://jimweller.com/jim-weller/jim/ java _proc _sched/#discussion ,2000

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AUTHOR PROFILES:

Mehdi Neshat was born in 1980.He received the B.Sc. degree in computer engineering from Azad University, Maybod, Iran, in 2006, the M.Sc. degree in Artificial Intelligence from the University of mashhad, in 2008 and is a member of the IEEE and the IEEE Computer Society He is with Islamic Azad University, Shirvan Branch, Faculty of Engineering, and computer Engineering Dept., Shirvan /Iran since 2007. His research interests are fuzzy logic, fuzzy systems, and fuzzy neural networks, particle swarm optimization, genetic algorithms, ant colony optimization, and other evolutionary computation techniques. He has publications and submissions in international conferences like applied soft computing, Applied Mathematical Modeling, Expert Systems with Applications, Fuzzy Sets & Systems, Computers in Industry Information Sciences, Mathematical & Computer Modeling.

Dr.Mehdi sargolzae was born in 1978.He received the B.Sc. degree in computer engineering from Ferdowsi University, mashhad, Iran, in 2005, the M.Sc. degree in computer engineering from the Amirkabir University of Tehran, in 2007 and Dr student in computer engineering from the Amsterdam University.

Adel Nadjaran Toosi is a PhD candidate in the Department of Computer Science and Software Engineering of the University of Melbourne, Australia. He's a member of Cloud Computing and **D**istributed **S**ystems (CLOUDS) Laboratory and my supervisor is Dr. Rajkumar Buyya. He used to be lecturer in Azad University of Mashhad. He received his B.S. degree in 2003 and M.S. degree in 2006 both in the department of Computer Software Engineering, Ferdowsi University of Mashhad, Iran. His research interests include Distributed Systems and Networking, Cloud Computing, Cloud Federation, Virtualization, Soft Computing and Fuzzy Systems. Currently I am involved in Inter-Cloud project which is a framework for federated Cloud Computing.

Ali Adeli was born in 1984.He received the B.Sc. degree in computer engineering from Islamic azad university of Shirvan ,in 2010 ,the M.Sc. degree in software engineering from the shiraz University of shiraz, in 2012.