Hindawi Publishing Corporation
Applied Computational Intelligence and Soft Computing
Volume 2014, Article ID 341957, 12 pages
http://dx.doi.org/10.1155/2014/341957

Research Article

2-Layered Architecture of Vague Logic Based Multilevel Queue Scheduler

Supriya Raheja,[1] Reena Dadhich,[2] and Smita Rajpal[3]

[1] *Department of Computer Science & Engineering, ITM University, Gurgaon, India*
[2] *Department of Computer Science, University of Kota, Near Kabir Circle, MBS Marg, Swami Vivekanand Nagar, Kota, Rajasthan 324 005, India*
[3] *Alpha Global IT, 1262 Don Mills Road, Toronto, ON, Canada M3B 2W7*

Correspondence should be addressed to Supriya Raheja; supriya.raheja@gmail.com

Received 28 July 2014; Revised 10 September 2014; Accepted 22 September 2014; Published 9 October 2014

Academic Editor: Baoding Liu

In operating system the decisions which CPU scheduler makes regarding the sequence and length of time the task may run are not easy ones, as the scheduler has only a limited amount of information about the tasks. A good scheduler should be fair, maximizes throughput, and minimizes response time of system. A scheduler with multilevel queue scheduling partitions the ready queue into multiple queues. While assigning priorities, higher level queues always get more priorities over lower level queues. Unfortunately, sometimes lower priority tasks get starved, as the scheduler assures that the lower priority tasks may be scheduled only after the higher priority tasks. While making decisions scheduler is concerned only with one factor, that is, priority, but ignores other factors which may affect the performance of the system. With this concern, we propose a 2-layered architecture of multilevel queue scheduler based on vague set theory (VMLQ). The VMLQ scheduler handles the impreciseness of data as well as improving the starvation problem of lower priority tasks. This work also optimizes the performance metrics and improves the response time of system. The performance is evaluated through simulation using MatLab. Simulation results prove that the VMLQ scheduler performs better than the classical multilevel queue scheduler and fuzzy based multilevel queue scheduler.

1. Introduction

In multitasking operating systems, multiple tasks need to be executed concurrently. Therefore, CPU scheduler plays a pivot role in operating system as it shares the CPU time among different tasks. For making the decision of scheduling next task for CPU, scheduler runs scheduling algorithm. Hence, the performance of system varies very much with scheduling algorithm used. Multilevel queue (MLQ) scheduling algorithm is among one of the preferable algorithms by OS designers [1, 2].

The kernel of operating system divides the CPU time among different queues depending on its requirement of I/O and CPU. But this share is fixed; it cannot be changed dynamically with variations in usage, since kernel is not aware of the exact parameters of task, like priority of task. However, in case of MLQ, priority plays a key role in decisions of scheduler. Recent evolutions in MLQ schedulers have contributed towards improvement of MLQ approach, but no significant enhancements to the approach which considers uncertainty factors [3]. There is one approach in literature that adapts the variations using fuzzy logic [4].

This paper concentrates on the dealing of uncertainty and impreciseness of task's parameters using another approach, that is, vague logic. Vague logic is an extended formation of fuzzy logic which becomes a dedicated tool to handle the imprecise information. With this aim, a vague inference system is designed inside the scheduler that deals with the uncertainty and impreciseness of tasks. This work also focuses on improving the performance of MLQ scheduler. We are introducing a new vague logic based MLQ scheduler which performs two main functions. First, it distributes the CPU time among different queues dynamically and adapts changes with the variations in usage. Second, it resolves the starvation problem of lower priority tasks by making decisions using vague based multilevel queue scheduling

algorithm. With these two functions the VMLFQ scheduler improves the response time of system as well which makes the system more responsive.

The paper is organized as follows. Section 2 discusses the introduction to MLQ scheduling algorithm and the related work with MLQ. Section 3 gives the brief idea about the vague set theory and how it is different from fuzzy set theory. In Section 4, we introduce the 2-layered architecture of VMLQ scheduler. Section 5, describes the simulation and results in detail. Finally Section 6 concludes the proposed work and discusses the future scope.

2. Related Work

MLQ scheduling algorithm has been created where different tasks can be classified into different groups. Scheduler organized the ready queue of system into multiple separate queues mainly for interactive and background tasks [5]. It is an extended form of priority scheduling, where scheduler assigns each task permanently to one queue based on its priority, type of task, and memory size. Each task has different response time requirements and may have different scheduling needs. Most operating systems, including Windows, Linux, and OS X, support a form of multilevel queues [6]. Let us consider two queues of two different types of tasks: interactive and batch tasks, in which interactive tasks are having higher priorities over batch tasks. In this type of system, scheduler schedules the tasks from those two queues as shown in Figure 1.

However, each queue has its own scheduling algorithm; usually interactive tasks run round robin scheduling technique and batch tasks run first come first serve [7]. These scheduling algorithms must be run between the queues. Scheduler dispatches the tasks from the highest priority queue and executes that task either preemptively or nonpreemptively. There are two different possibilities to schedule the tasks between the queues. In fixed priority scheduling all the tasks from the higher priority queue are executed first. The lower priority tasks from the batch queue will not be executed unless the higher priority queue will be empty. In this type of scheduling, there will be chances of occurrence of starvation [4].

However in the second type, each queue gets a fixed amount of CPU time iteratively which is further distributed amongst its tasks. In the above supposed case, the ready queue with n is partitioned into 2 smaller ready subqueues, Q1 and Q2, where Q1 have the same priority tasks from 1 to j and Q2 have from $J + 1$ to n. In preemptive priority scheduling for the multiqueue technique, all tasks from 1 to j in ready subqueue Q1 will be completed before any task is run from ready subqueue Q2 through $j + 1$ to n. Within the subqueues the CPU may be allocated using any techniques, but mainly the high priority Q1 using round robin and the low priority queue, that is, Q2, using FCFS. There are different static techniques available to share the CPU time across the subqueues. For example, each subqueue can get the fraction of CPU time; suppose that 100 seconds of CPU time can be partitioned statically as 50 seconds among all the tasks or 80% for Q1 or 20% for Q2 and so on [1].

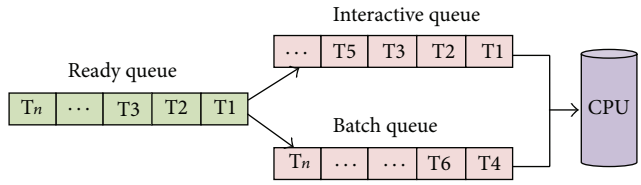

Figure 1: Multilevel queue technique.

As we have mentioned earlier, there is very limited literature available for MLQ scheduling algorithm. Shukla et al. have presented a MLQ scheduling algorithm using Markov chain model [8]. This scheduler can perform random number of movements based on different processing states and waiting states. By adding queue priorities in the scheduling policy intelligently, this system can perform better. Nasser et al. have introduced a new packet scheduling algorithm called dynamic multilevel queue scheduling for wireless networks [9]. Nasser et al. proposed an algorithm in which the queue sizes can vary based on the application requirement [9]. Shah et al. have proposed multilevel hybrid scheduling algorithm for optimum utilization of processors in grid environment [10]. But any of this work did not consider uncertainty and impreciseness of task. For example, the task with lower priority with small CPU time has always been assigned to lower level queues using these techniques; but handling the impreciseness dynamically rather than fixing the queues on the basis of priority, this task may be assigned to higher queues and can improve the performance of scheduler.

Professor Zadeh had designed a theory for dealing with imprecise data that provides the scope of further enhancements [11–14]. Different scheduling algorithms are available in literature using fuzzy set theory [15–17]. Chahar and Raheja have explored this scope and introduced a fuzzy based MLQ scheduling algorithm. We are calling this algorithm FMLQ scheduling algorithm [4]. They have introduced a dynamic approach to share the CPU time among different queues and provided a fuzzy based solution over fixed amount of CPU time distribution. Undoubtedly, FMLQ scheduling algorithm brought a solution for imprecise information and also improved the performance of system. Our present work explores the scope of further enhancement in development of techniques dealing with uncertainty and impreciseness in a much better way. Therefore, this work uses the concept of vague set theory which is an extended form of fuzzy set theory.

The significant aspect of VMLQ scheduler is that it provides a dynamic share of CPU time to queues. It not only considers the prime factor of MLQ scheduling priority but also considers the associated uncertainty and impreciseness. The VMLQ scheduler improves the performance of system over the MLQ scheduling and FMLQ scheduling algorithm.

In the next section, we will describe the core part of our work, that is, vague set theory.

3. Vague Set Theory

In 1965, Professor Zadeh has revolutionized the theory of logics by introducing fuzzy set theory. In fuzzy set theory every object has single degree of membership in between 0 and 1. Let X be the universe of discourse, where $X = \{x_1, x_2, \ldots, x_n\}$.

Definition 1 (fuzzy set). A fuzzy set F in X is a set of ordered pairs $\{x, \mu_F(x) : x \in X\}$, where $\mu_F(x)$ is the degree of membership of element x in the set F. The larger the value of $\mu_F(x)$means is, the more the element x belongs to the set F [11].

Well-known extensions of fuzzy set theory have been proposed such as type-2 fuzzy sets having membership functions that map from a set X to a type-1 fuzzy set on the interval $[0, 1]$. Such type-2 sets are φ-fuzzy sets, which map from X to the set of closed intervals in $[0, 1]$. Such an interval-valued fuzzy set is characterized by a membership function $\mu_F(x)$, $x \in X$, which assigns to each element a grade of membership over a continuous interval of real numbers in the range $[0, 1]$, instead of a single value. It also assigns a grade of membership which is a subinterval of $[0, 1]$ to each element. This subinterval keeps track of both evidences: evidence for favour and evidence for opposition. Quinlan introduced two values, $t(P)$ and $f(P)$, characterizing a proposition P. $t(P)$ is the greatest lower bound on the probability of P derived from the evidence for P and $f(P)$ is the greatest lower bound on $\sim P$ derived from the evidence against P.

In 1993, Professors Gau and Buehrer had extended this single membership concept with the two membership concepts, true-membership function $t_F(x)$ and false-membership function $f_F(x)$, to record the lower bounds on $\mu_F(x)$. They addressed the two membership degrees, degree of favour and degree of opposition individually rather than single membership value $\mu_F(x)$ as in fuzzy set theory. These membership values even can process the two evidences at the same time. They had investigated that single membership degree cannot give more accuracy. Gau and Buehrer have introduced the interval based theory, that is, vague set theory, over single membership theory. The two defined membership functions are true-membership function t_F and false-membership function f_F which generalize the μ_F of fuzzy set. For $\forall x$, $t_F(x) \leq \mu_F(x) \leq f_F(x)$ [18].

Definition 2 (vague set). A vague set V in X is characterized by a true-membership function t_V and a false-membership function f_V:

$$t_V : X \longrightarrow [0, 1], \qquad f_V : X \longrightarrow [0, 1], \tag{1}$$

where $t_V(x)$ is a lower bound on the grade of membership of x derived from the evidence for x and $f_V(x)$ is a lower bound on the negation of x derived from the evidence against x. There is a higher order check on the two independent membership values that total amount cannot exceed 1; that is, $t_V(x) + f_V(x) \leq 1$. Thus the grade of membership of x in the vague set V is bounded by a subinterval $[t_V(x), 1 - f_V(x)]$ of $[0, 1]$. This indicates that if the fuzzy grade of membership is $\mu_F(x)$, then $t_V(x) \leq \mu_F(x) \leq 1 - f_V(x)$ [18].

Definition 3 (vague value). The interval $[t_V(x), 1 - f_V(x)]$ is called the "vague value" of x in V. The vague set V is written as

$$A = \langle x, [t_V(x), f_V(x)] \rangle : x \in X. \tag{2}$$

Let us consider a vague value $[0.6, 0.8]$ in a vague set V. Here, $t_V = 0.6$, $1 - f_V = 0.8$, and 0.2 is the hesitated value that does not belong to either the true value or the false value. The total scope of these values is between 0 and 1 [19, 20].

3.1. Deviation of Fuzzy Set towards Vague Set. Let X be a universe of discourse; suppose collection of static priority of tasks of OS. Let V be a vague set of all "medium priority tasks" of X, and let F be a fuzzy set of all "medium priority tasks" of X. Suppose an expert e1 advises degree of membership $\mu_F(x)$ for an object x in fuzzy set F by his expertise. Whereas at the other side, another expert e2 advises two degrees of memberships $t_V(x)$ and $f_V(x)$ for the same object x in vague set V by his expertise. Both experts e1 and e2 have their limitation of sensing power, assessment capability, and working ability with real life situations. In case of fuzzy set F, there is no further check on degree of membership $\mu_F(x)$. However, an expert e2 advised degree of memberships independently but can have a further check by maintaining the constraint $t_V(x) + f_V(x) \leq 1$ [21].

4. Vague Logic Based Multilevel Queue CPU Scheduler

VMLQ scheduler has the capability of observing, learning, and holding information about ready tasks. This learning power makes the scheduler capable enough to respond dynamically, hence assigning different CPU time among queues at different situation. Suppose at time $t1$ that the CPU time for Q1 is 65% and for Q2 is the remaining 35%, though at time $t2$, 75% CPU time can be given to Q1 and the remaining 25% to Q2. Traditional scheduler assigns the fixed share to queues [1]. But our scheduler responds dynamically depending on the current state of ready queue. VMLQ scheduler performs mainly two tasks. First, it distributes the CPU time among different queues dynamically and adapts changes with the variations in usage. Second, it resolves the starvation problem of lower priority tasks by making decisions using vague based multilevel queue scheduling algorithm.

VMLQ scheduler has two layers as shown in Figure 2. Lower layer runs vague inference system and upper layer runs scheduling algorithm. In the next sections we discuss the working of these layers in detail

4.1. Vague Inference System for VMLQ Scheduler. A vague inference diagram expresses all the steps from vaguification to devaguification [21] as shown in Figure 3. Vague inference system (VIS) maps the crisp inputs into crisp outputs using vague set theory [22]. This mapping provides a base for the

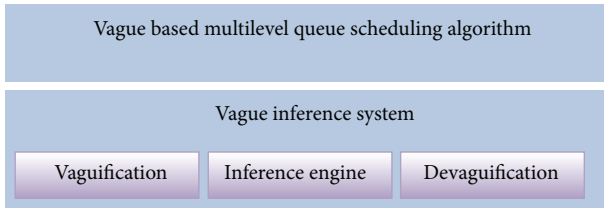

Figure 2: 2-layered architecture of VMLQ scheduler.

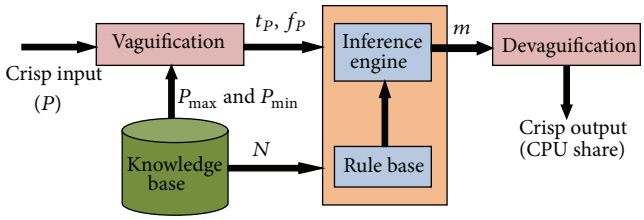

Figure 3: Vague inference system for VMLQ scheduler.

decisions to be made. We considered a VIS for a uniprocessor system that supports N number of tasks.

Vaguification. Vaguification is a mathematical procedure which determines the degree of belongingness of input in form of membership functions. It takes the input v from the universe of discourse X in crisp form and converts it into appropriate vague data $[t_V, 1 - f_V]$, where t_V represents true-membership value and f_V represents false-membership value.

Let us assume universe of discourse $P = \{P_1, P_2, \ldots, P_N\}$, where an element of P is denoted by P_i and represents the user priority of ith task.

In the process of vaguification, membership functions t_P and f_P which are defined in (3) are applied on crisp input P_i so that the degree of truth and degree of false $\forall i$ can be determined. It handles the impreciseness by considering the membership degree with respect to maximum and minimum values of input, so that imprecise value will be equally dispersed among all tasks. Hence, this does not affect the final decision. Consider

$$\begin{aligned} \forall i, \quad t_{Pi} &= \frac{P_{\max} - P_i}{P_{\max} + P_{\min} + 1}, \\ \forall i, \quad f_{Pi} &= \operatorname{sqrt}\left(\frac{P_i - P_{\min}}{P_{\max} + P_{\min} + 1}\right). \end{aligned} \tag{3}$$

The defined membership functions should follow the four axioms and all axioms are represented in Figure 4.

Axiom 1. $t_P \leq 1$.

Axiom 2. $f_P \leq 1$.

Axiom 3. $0 \leq t_P \leq 1 - f_P \leq 1$.

Axiom 4. $0 \leq t_P + f_P \leq 1$.

Vague data $\forall i$ can be represented as

$$\begin{aligned} &[t_{Pi}, 1 - f_{Pi}] \\ &= \left[\frac{P_{\max} - P_i}{P_{\max} + P_{\min} + 1}, 1 - \frac{P_i - P_{\min}}{P_{\max} + P_{\min} + 1}\right]. \end{aligned} \tag{4}$$

Knowledge Base. It acts as the storage for the ready tasks. All the information associated with tasks like burst time, user priorities, arrival time, maximum available priority, minimum available priority, number of active tasks, and so forth are stored in the system. Consider

$$\begin{aligned} P_{\max} &= \max\{P_1, P_2, \ldots, P_N\}, \\ P_{\min} &= \min\{P_1, P_2, \ldots, P_N\}. \end{aligned} \tag{5}$$

Rule Base. Rule base mainly consists of *if-then* rules. In our rule base, we have defined two rules based on the number of tasks ready in the system. Consider

$$\begin{aligned} &\text{if } N < 5 \text{ then } S_i = N * 10, \\ &\text{if } N \geq 5 \text{ then } S_i = N * 5. \end{aligned} \tag{6}$$

Vague Inference Engine. It extracts the information $\forall i$, t_P, and $1 - f_P$ from the vaguification process and S from rule base. After vaguifying the data, system knows the degree of favour and degree of opposition. Based on the degrees, vague inference engine evaluates the rules defined in rule base. Afterward, based on the evaluation, system computes the m as follows:

$$\forall i, \quad m_i = S_i * (t_{Pi} + 1 - f_{Pi}). \tag{7}$$

This phase results in fuzzy value which is assigned to each task. The inference process should follow Axiom 5. The multiple inputs can be passed to the inference process.

Axiom 5. $0 \leq m_i \leq 1$.

Devaguification. The purpose of devaguification is to find the crisp value of the output.Generally, the required output is a single number. However, the vague inference engine produces the multiple fuzzy values with respect to input variables. Devaguification process converts the multiple output values into a single number. Hence, the task with the maximum fuzzy value m is chosen for the output as shown in Figure 5.

The crisp value for the output *CPU share* is computed by multiplying the maximum fuzzy value by 100 as follows:

$$\text{CPU}_{\text{share}} = 100 * \max\{m_1, m_2, \ldots, m_N\}. \tag{8}$$

4.2. Vague Based Multilevel Queue Scheduling Algorithm. In VMLQ scheduling algorithm the ready queue is divided into two subqueues, Q1 and Q2, as shown in Figure 6. Q1 is

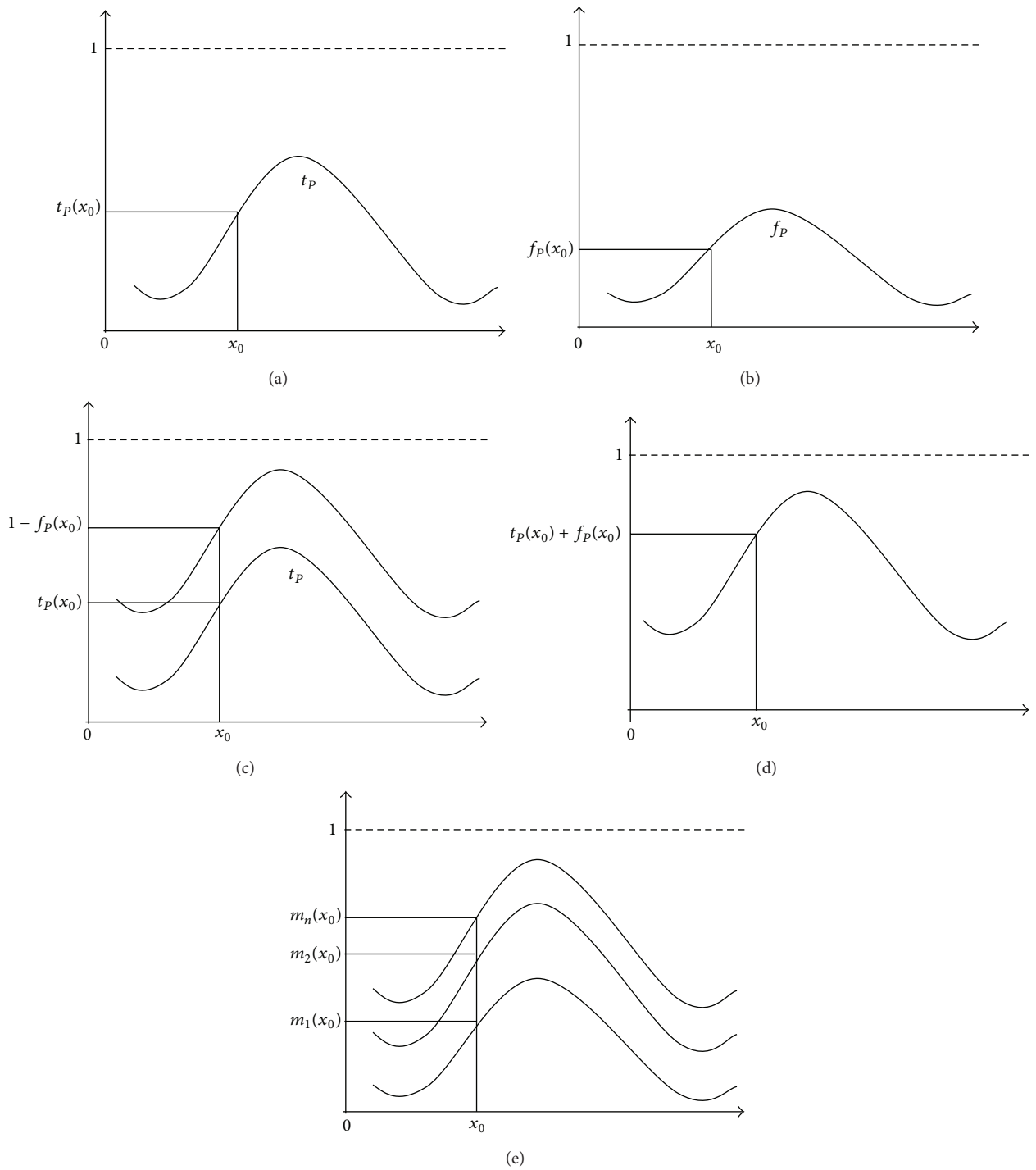

Figure 4: (a) Axiom 1, (b) Axiom 2, (c) Axiom 3, (d) Axiom 4, and (e) Axiom 5.

holding interactive tasks whereas Q2 is holding background tasks. Since interactive tasks are the highest priority tasks among all tasks [3], therefore this induces higher priority for Q1 over Q2. The tasks are permanently assigned to each queue as in MLQ scheduling algorithm. So, task from Q1 cannot be moved to Q2 and vice versa.

The CPU time is shared dynamically among Q1 and Q2. The CPU time for Q1 (Q1_time) is calculated using the vague inference system as discussed in Section 4.1. The CPU time for Q2 (Q2_time) is calculated using the CPU share of Q1. The tasks of Q1 are dispatched with CPU only for Q1_time. However, tasks from Q2 get the CPU only when either Q1_time becomes zero or Q1 becomes empty. Both queues have their own scheduling algorithm, as both interactive and background tasks need different response.

Each queue has its own scheduling algorithm, because both types of tasks have their different response time requirements. Interactive tasks in Q1 are scheduled using round

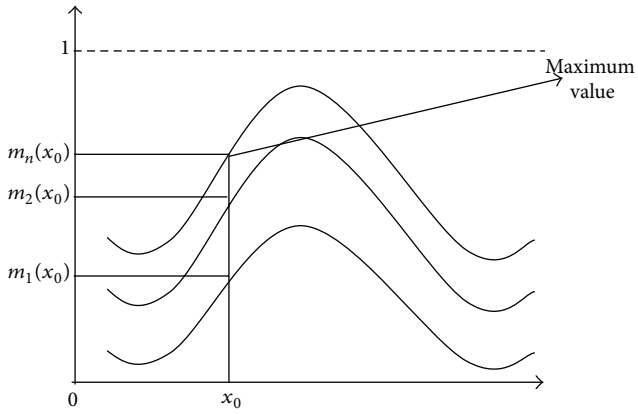

Figure 5: Task with maximum fuzzy value.

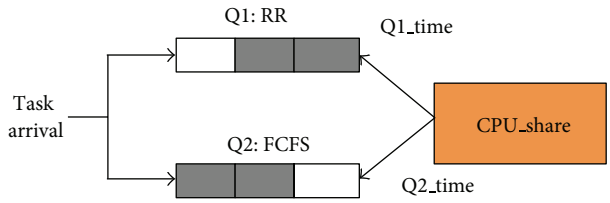

Figure 6: Schematic view of proposed system.

robin (RR) scheduling algorithm [23, 24]; it means that each task is dispatched to CPU only for given time quantum. After the expiry of each time quantum, the current running task will preempt and the next task from Q1 will be scheduled with CPU. This process of switching CPU from one task to another task is called context switch which is totally an overhead for system, since no useful work is performed by the system during context switching [16]. Sometimes burst time of task is remaining in between 0.1 and 0.5 sec when time quantum expires, which increases waiting time of task as well as the number of context switches for system. For reducing the number of context switches, VMLFQ scheduler will not preempt the task if the remaining burst time of that task is less than 0.5 sec.

VMLQ scheduler uses new first come first serve (NFCFS) scheduling algorithm for all tasks of Q2. NFCFS scheduling algorithm schedules the task as soon as it arrives in the system. However, traditional FCFS scheduling algorithm schedules the next task only after completing the execution of the first task. In interactive environment, response time is the major factor to measure performance of scheduler. FCFS is nonpreemptive type of scheduling algorithm which increases the response time as well as waiting time for each task [1], whereas NFCFS is a preemptive scheduling algorithm which improves the response time as compared to FCFS.

4.2.1. Algorithm

Step 1. Classify task into interactive and background tasks based on burst time.

Step 2. Assign I/O bound tasks to Q1 and CPU bound tasks to Q2:

$N1$ = number of tasks in Q1,

$N2$ = number of tasks in Q2.

Step 3. Sort Q1 in ascending order based on burst time.

Step 4. Calculate the percentage of CPU time allocation for Q1:

Q1_time = $\text{CPU}_{\text{share}}$ (using (8)).

Step 5. Calculate the CPU time for Q2 (Q2_time):

Q2_time = 100 − Q1_time).

Step 6. Schedule the tasks from Q1 using RR scheduling algorithm:

Time quantum (TQ) = Q1_time/$N1$.

Step 7. If (Q1_time == 0 ‖ $N1$ == 0) then

schedule tasks from Q2 using NFCFS.

Step 8. After each cycle repeat steps from 1 to 8 until ($N2$ == 0).

5. Simulation and Results

The performance metrics must be chosen carefully as they reflect the characteristics of system. The metrics chosen are as follows. Response time is defined as the amount of time a system takes to respond to the user input. It is one of the most important factors in CPU scheduling algorithms. Another metric which is considered in our work is waiting time, which is defined as the amount of time tasks wait in the ready queue for CPU. In our work, total waiting time is the waiting time of task in both higher priority queue and lower priority queue. Next metric is turnaround time, total time consumed by task from its submission to its completion. The last metric is normalized turnaround time, relative delay of the task [1]. The reduction in values of all these parameters represents improvement in the performance of system. We have implemented the proposed VIS-VMLQ using MatLab. To compare the performance of proposed algorithm with the traditional MLQ and FMLQ, the following procedure has been taken.

For each $N \in [4, \ldots, 20]$, multiple sets of random tasks were considered. Then these three algorithms were used to schedule the tasks. Results show that VMLQ approach performs better than other approaches. We are presenting the three scheduling algorithms MLQ, FMLQ, and VMLQ with the help of sample task set [] as given in Table 1. Total 16 tasks are considered, out of these 11 tasks are classified as interactive/IO bound tasks and remaining 5 as background/CPU bound tasks. I/O bound tasks are represented by T and CPU bound tasks by C. Assume all measures in seconds. Suppose constant CPU cycle time for each algorithm as 20 seconds. The scheduling sequence of each algorithm is illustrated with the help of Gantt charts.

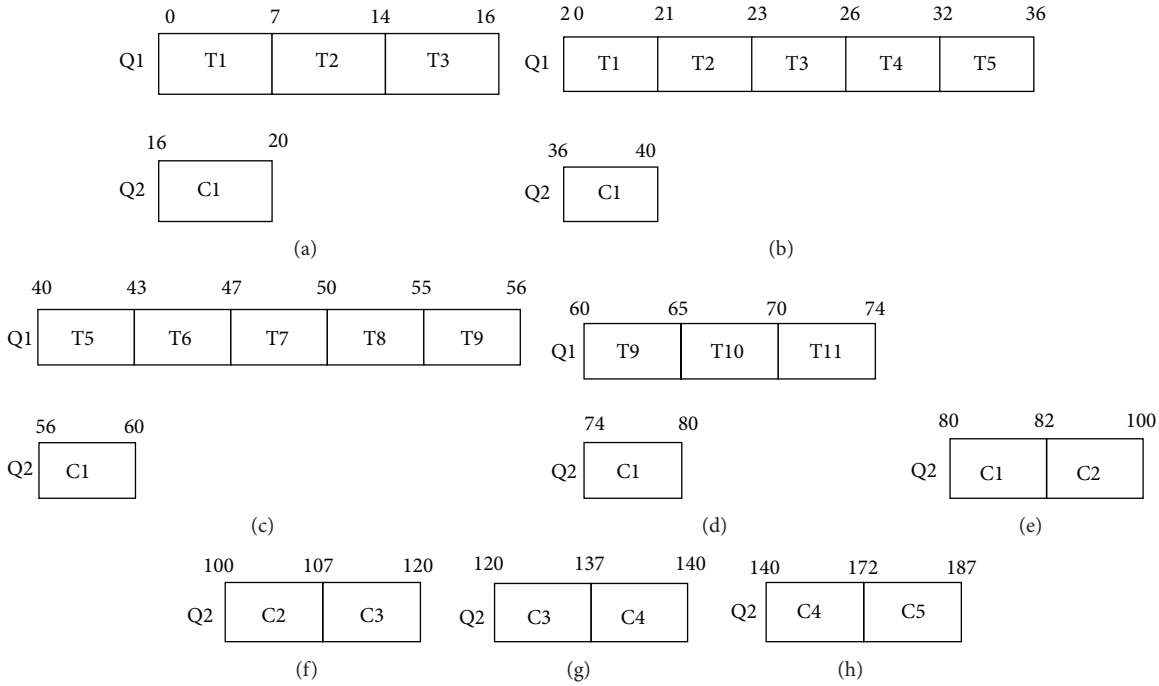

Figure 7: (a) 1st cycle of CPU (MLQ), (b) 2nd cycle of CPU (MLQ), (c) 3rd cycle of CPU (MLQ), (d) 4th cycle of CPU (MLQ), (e) 5th cycle of CPU (MLQ), (f) 6th cycle of CPU (MLQ), (g) 7th cycle of CPU (MLQ), and (h) 8th cycle of CPU (MLQ).

Table 1: Sample task set.

I/O bound tasks				CPU bound tasks			
Task	Arrival time	Priority	Burst time	Task	Arrival time	Priority	Burst time
T1	0	9	8	C1	0	3	20
T2	0	8	9	C2	0	2	25
T3	0	7	5	C3	30	3	30
T4	21	6	6	C4	35	2	35
T5	25	5	7	C5	40	3	15
T6	38	7	4				
T7	40	8	3				
T8	45	6	5				
T9	55	6	6				
T10	57	8	5				
T11	60	5	4				

Figure 7 shows the scheduling sequence of task set given in Table 1 for MLQ scheduling algorithm. With the MLQ, the tasks from Q1 are scheduled using RR algorithm and from Q2 using FCFS algorithm. Let us consider the length of static time quantum as 7 s. As discussed in Section 2, assign fixed 80% (16 s) CPU share to Q1 and the remaining 20% (4 s) to Q2. During first cycle the tasks T1, T2, and T3 are scheduled until 16 s and C1 is scheduled for 4 s as shown in Figure 7(a).

Till the end of the 2nd cycle tasks T1, T2, T3, and T4 finish their execution as shown in Figure 7(b). Likewise the tasks from Q1 and Q2 are scheduled in the 3rd and 4th cycle as shown in Figures 7(c) and 7(d). After the 4th cycle, Q1 becomes empty. Now the complete cycle of 20 s is assigned to Q2. The scheduling sequence of Q2 tasks is shown in Figures 7(e), 7(f), 7(g), and 7(h), respectively.

Figure 8 shows the schedule sequence with FMLQ scheduling algorithm of the same task set. After applying FMLQ, it returns the size of time quantum for first cycle as 5 s and CPU share for Q1 as 65% (13 s) whereas the remaining 35% (7 s) for Q2. The shortest task of Q1 is scheduled first to CPU as shown in Figure 8(a). Tasks from Q2 are scheduled as soon as they arrive in the queue; hence C1 and C2 both are scheduled for 3.5 s each.

During the 2nd cycle, FMLQ gives the CPU share to Q1 as 85% (14 s) and to Q2 as 15% (6 s). Each task in Q1 is scheduled for 4 s, time quantum for Q1 as shown in Figure 8(b). Whereas

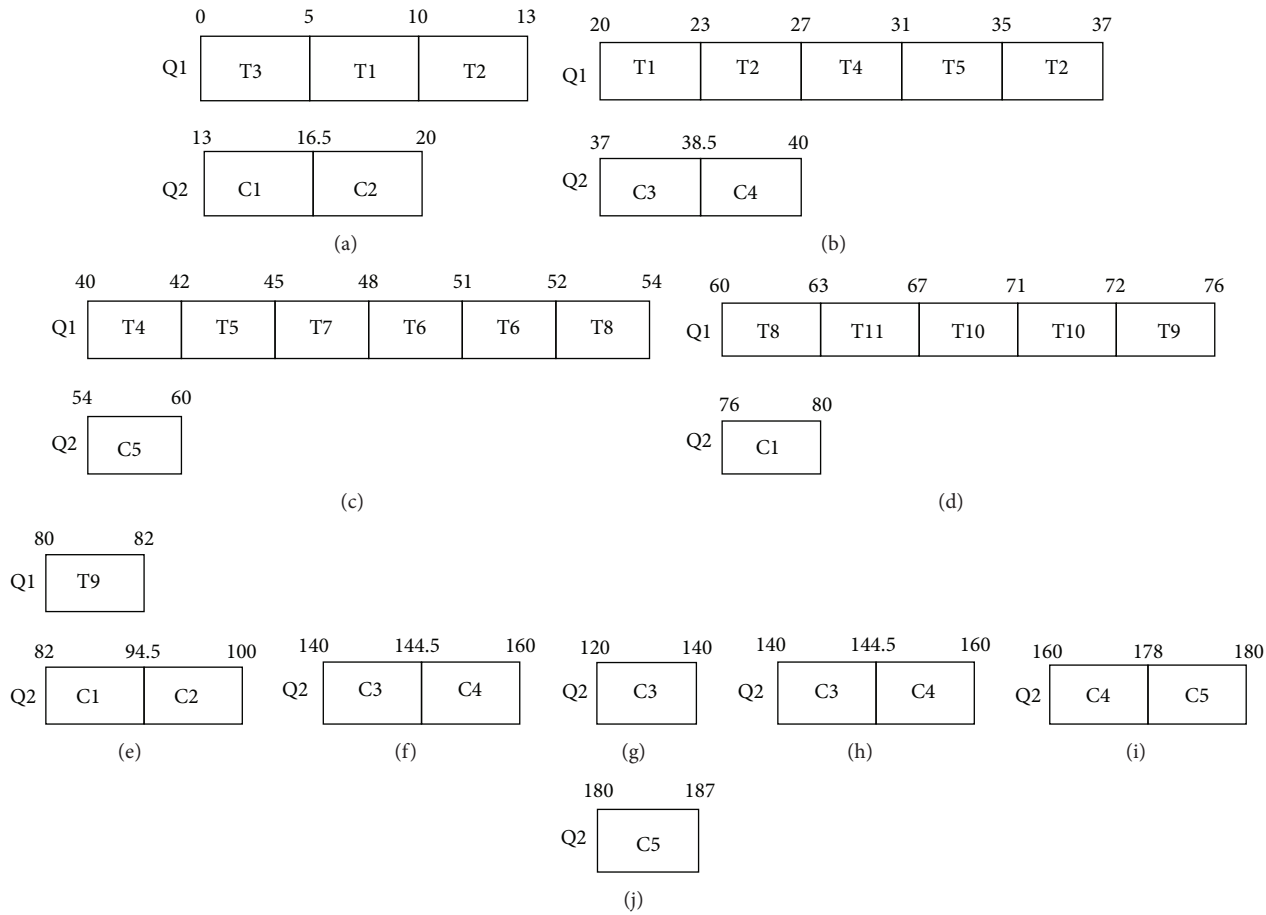

Figure 8: (a) 1st cycle of CPU (FMLQ), (b) 2nd cycle of CPU (FMLQ), (c) 3rd cycle of CPU (FMLQ), (d) 4th cycle of CPU (FMLQ), (e) 5th cycle of CPU (FMLQ), (f) 6th cycle of CPU (FMLQ), (g) 7th cycle of CPU (FMLQ), (h) 8th cycle of CPU (FMLQ), (i) 9th cycle of CPU (FMLQ), and (j) 10th cycle of CPU (FMLQ).

for the third and fourth cycle, FMLQ returns size of time quantum for Q1 as 3 s and 4 s, respectively. Likewise, it returns the CPU share as 70% and 80%, respectively. In FMLQ, if the remaining burst time is less than or equal to 1, the task will not preempt rather it finishes its execution as indicated in Figure 8(c).

Up to the third cycle, all the CPU bound processes have arrived in Q2 and scheduled once to CPU. In the fourth cycle task C1 has resumed its execution for 4 s as shown in Figure 8(d).

All the I/O bound tasks have finished their execution up to the 5th cycle as shown in Figure 8(e). Afterwards the entire CPU share has been assigned to Q2. We can see the complete scheduling from Figures 8(f)–8(j).

Figure 9 illustrates the scheduling order of each task using VMLQ on the same task set. For the first cycle, VIS return 70% (15.8 s) CPU share for Q1 as discussed in Section 4 and 30% (4.2 s) for Q2. VMLQ algorithm return time quantum for Q1 as 5.4 s. Both tasks, T1 and T2, are scheduled for 5.4 s as shown in Figure 9(a).

Task in Q2 is dispatched in the same way as discussed in FMLQ. Both tasks, C1 and C2, are scheduled for 2.1 s. For the second cycle, VMLQ returns CPU share for Q1 as 69% (12.6 s) and time quantum as 3.2 s whereas it returns CPU share for Q2 as 31% as shown in Figure 9(b).

For the third cycle, Q1 receives 15.3 s CPU time and Q2 gets 4.7 s of CPU cycle as shown in Figure 9(c). The tasks T4 and T5 have resumed and finished their execution. Similarly, the CPU time has been computed for the 4th and 5th cycle for Q1 as 13.9 s and 14.4 s, respectively. Up to the 4th cycle, all the tasks from Q1 have dispatched once with CPU as shown in Figure 9(d). During the 5th cycle Q1 becomes empty as shown in Figure 9(e). From the 6th cycle onwards all the CPU time is given to CPU bound tasks as shown in Figure 9(f). We have represented all cycles after the 5th one in a single Gantt chart.

In addition to the distribution of CPU time among Q1 and Q2, we are also interested in the performance improvement of the system. So, based on the above discussed Gantt charts, we have calculated the performance metrics, response time, waiting time, turnaround time, and normalized turnaround time for all tasks. The respective outputs for all algorithms are shown in Figures 10, 11, 12, and 13, respectively.

Then we compared the average waiting time, average response time, and average normalized turnaround time for

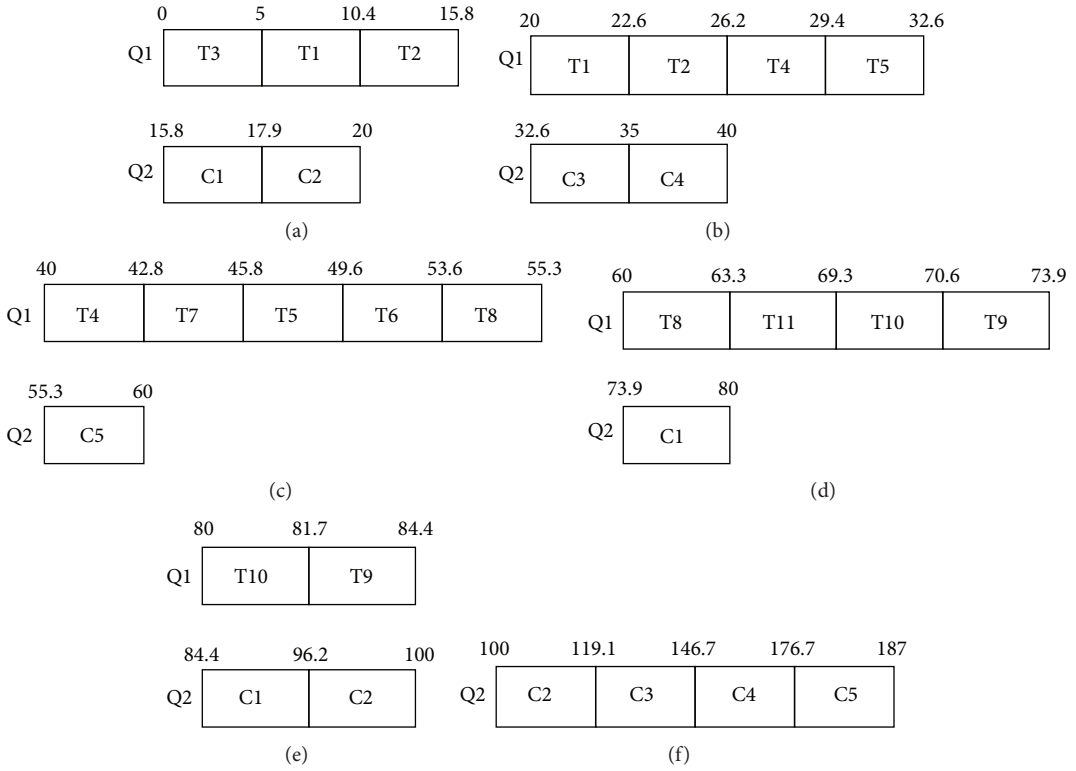

Figure 9: (a) 1st cycle of CPU (VMLQ), (b) 2nd cycle of CPU (VMLQ), (c) 3rd cycle of CPU (VMLQ), (d) 4th cycle of CPU (VMLQ), (e) 5th cycle of CPU (VMLQ), and (f) 6th–10th cycles of CPU (VMLQ).

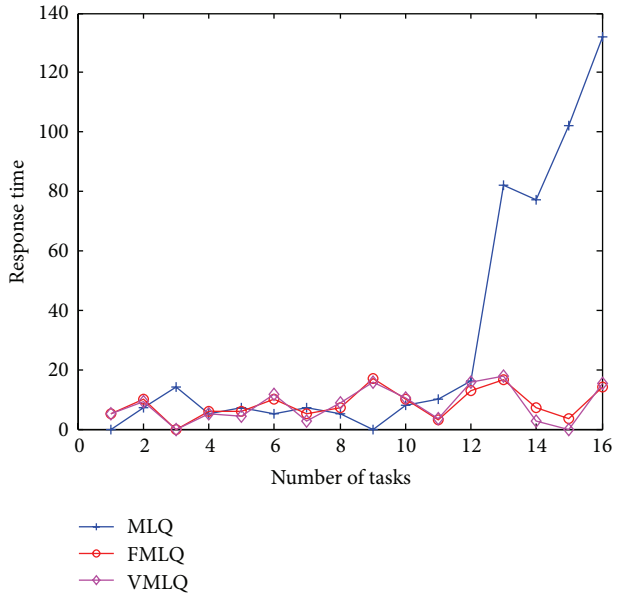

Figure 10: Response time (sample task set).

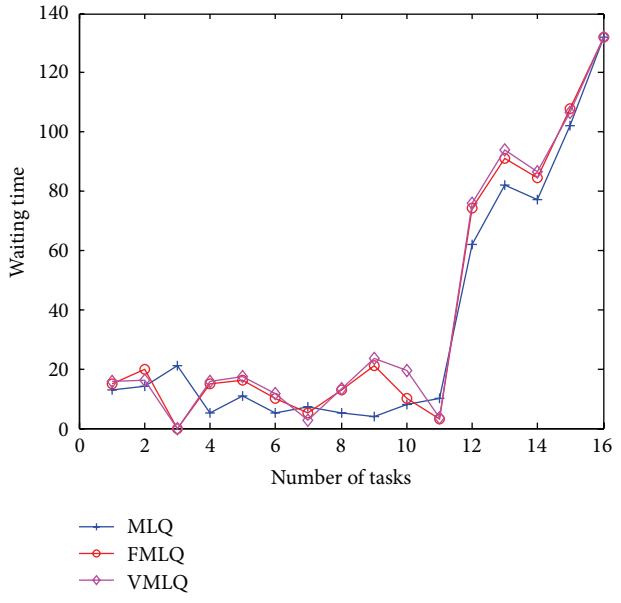

Figure 11: Waiting time (sample task set).

all three algorithms as shown in Figure 14. On x-axis, "1" represents MLQ scheduling algorithm, "2" represents FMLQ scheduling, and "3" represents VMLQ scheduling algorithm.

One can notice from Figure 14 that there is large dropoff in value of average response time from MLQ to FMLQ and from FMLQ to VMLQ. Dropoff represents the improvement in performance. However, there is slight deviation in average waiting time, average turnaround time, and average normalized turnaround time that is acceptable, as response time is

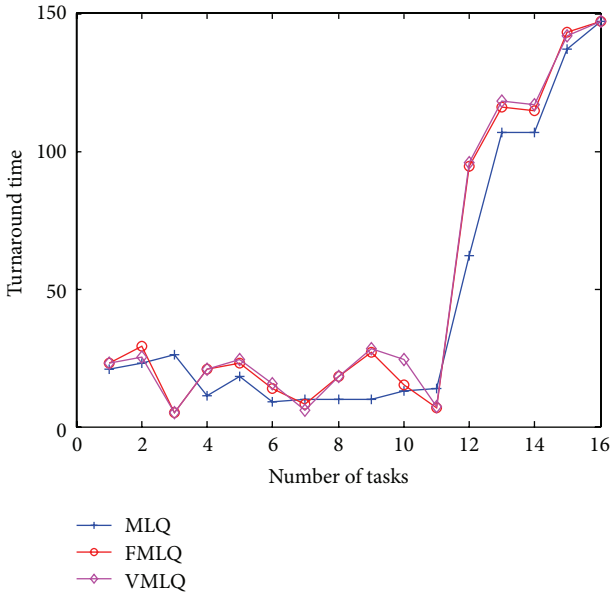

Figure 12: Turnaround time (sample task set).

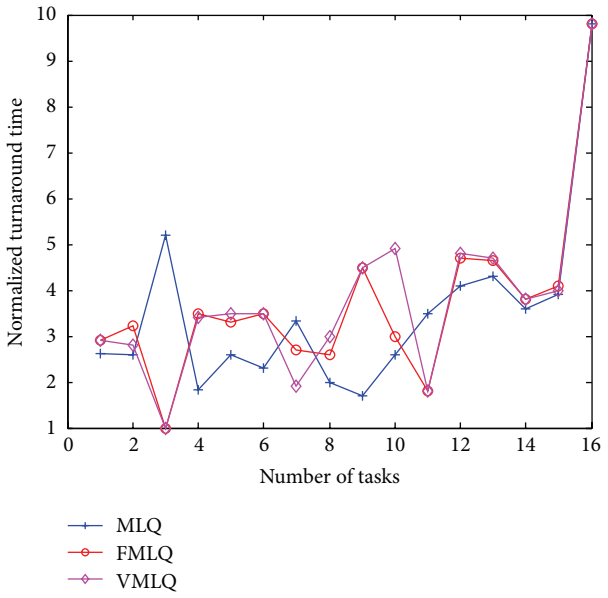

Figure 13: Normalized turnaround time (sample task set).

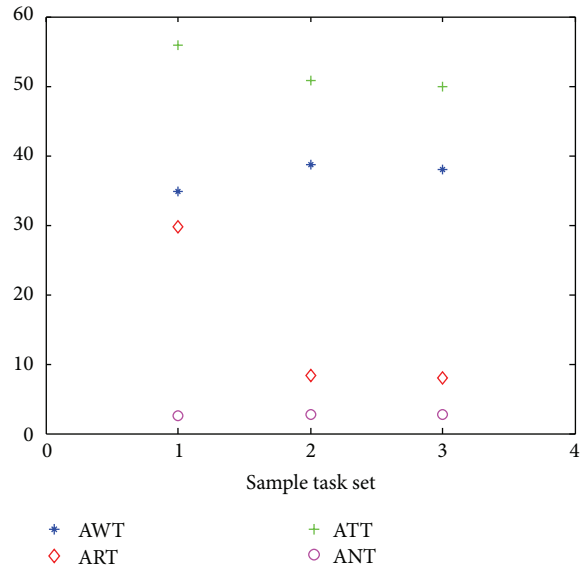

Figure 14: Results (sample task set).

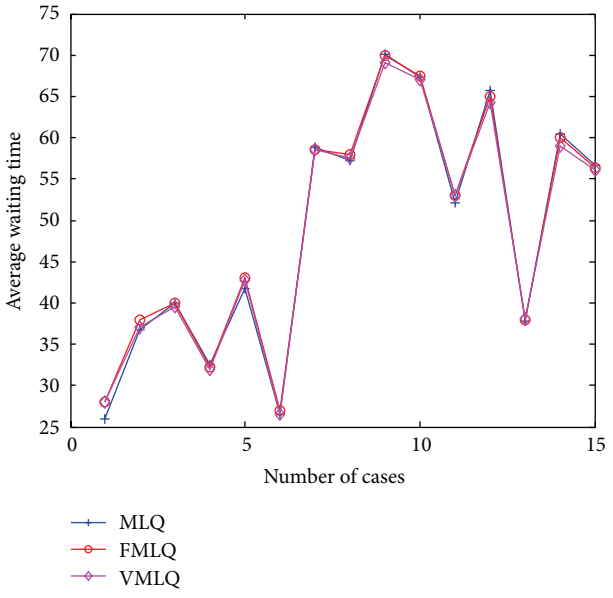

Figure 15: Average waiting time.

the major considerable factor in multitasking or interactive systems.

We have repeated this process randomly over multiple tasks set. Further we have calculated all permformance metics for each task set. Finally we have compared the results of all three algorithms as shown in Figures 15, 16, 17, and 18.

Result in Figures 15, 16, 17, and 18 indicates that VMLQ has an excellent performance over MLQ and FMLQ. Among all algorithms, VMLQ has the lowest average response time. It is understandable that performance improvement in relation to response time can slightly increase the other factors including average waiting time and average normalized turnaround time as we have discussed in our work. VMLF scheduling algorithm has better performance mainly by two reasons.

The proposed architecture of VMLQ scheduler is very effective in highly dynamic environment where CPU time is shared dynamically among multiple queues and further among tasks in each queue. It deals with impreciseness in more better way as compared to fuzzy set theory. In addition to sharing of CPU time, decreased response time improves the starvation problem at the lower priority queues since at least once the task has been scheduled to CPU as soon as it arrives to the system.

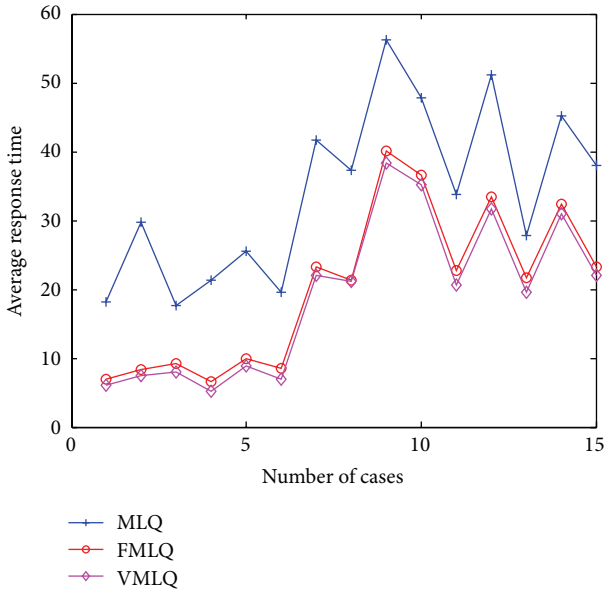

Figure 16: Average response time.

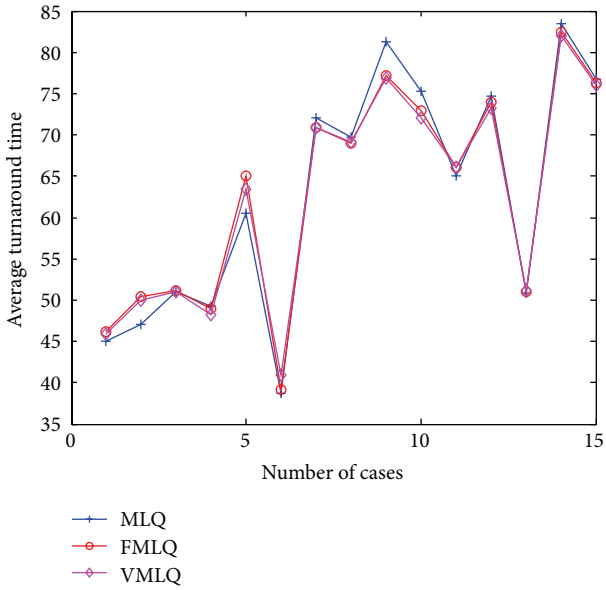

Figure 17: Average turnaround time.

6. Conclusion

The 2-layered architecture for VMLQ scheduler introduced in this paper offers solutions for the two basic problems over classic MLQ algorithm and FMLQ algorithm. Firstly, it is more appropriate for dynamical allocation of CPU time among multiple queues because of its capability to approximate the input parameters, user priority, and number of tasks. Further, it improves the average response time of system and consequently solves the starvation in the lower priority queues. The proposed methodology implements the vague inference system using MatLab that takes the crisp input priority and converts it into vague data to handle

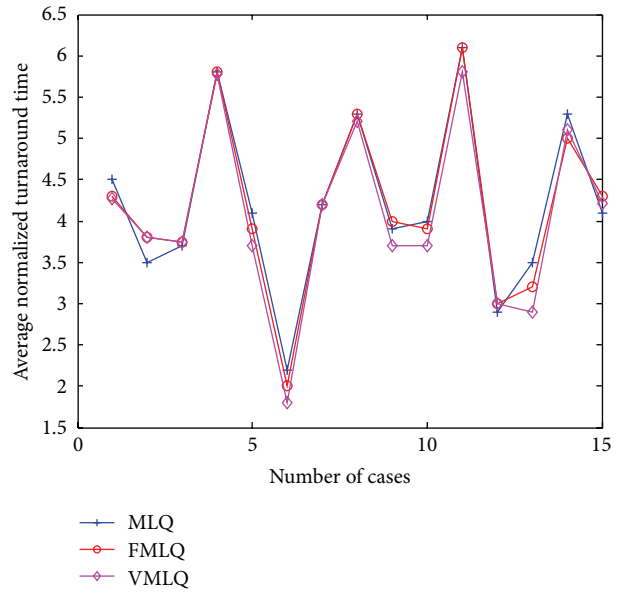

Figure 18: Average normalized turnaround time.

the impreciseness of data. Simulation results in Section 5 demonstrate that this algorithm excellently improves the performance of classic MLQ scheduling algorithm and the FMLQ scheduling algorithm. In future, VMLQ scheduling algorithm can be extended for more partitions of ready queue.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] A. Silberschatz and P. B. Galvin, *Operating System Concepts*, John Wiley & Sons, New Delhi, India, 6th edition, 2008.

[2] W. Stallings, *Operating Systems Internal and Design Principles*, Pearson Education, 6th edition, 2006.

[3] M. V. Panduranga Rao and K. C. Shet, "Analysis of new multi-level feedback queue scheduler for real time kernel," *International Journal of Computational Cognition*, vol. 8, pp. 5–16, 2010.

[4] V. Chahar and S. Raheja, "Fuzzy based multilevel queue scheduling algorithm," in *Proceedings of the 2nd International Conference on Advances in Computing, Communications and Informatics (ICACCI '13)*, pp. 115–120, August 2013.

[5] S. Lim and S.-B. Cho, "Intelligent OS process scheduling using fuzzy inference with user models," in *New Trends in Applied Artificial Intelligence*, vol. 4570 of *Lecture Notes in Computer Science*, pp. 725–734, Springer, Berlin, Germany, 2007.

[6] Kadhim and K. M. AI-Aubidy, *Design and Evaluation of a Fuzzy Based CPU Scheduling Algorithm*, Springer, Berlin, Germany, 2011.

[7] N. Bin, D. Jianqiang, X. Guoliang, L. Hongning, Y. Riyue, and W. Quan, "A new operating system scheduling algorithm," in *Advanced Research on Electronic Commerce, Web Application, and Communication*, vol. 143, pp. 92–96, Springer, 2011.

[8] D. Shukla, S. Ojha, and S. Jain, "Data model approach and Markov Chain based analysis of multi- level queue scheduling," *Journal of Applied Computer Science Mathematics*, vol. 8, no. 4, pp. 50–56, 2010.

[9] N. Nasser, L. Karim, and T. Taleb, "Dynamic multilevel priority packet scheduling scheme for wireless sensor network," *IEEE Transactions on Wireless Communications*, vol. 12, no. 4, pp. 1448–1459, 2013.

[10] S. N. M. Shah, A. K. B. Mahmood, and A. Oxley, "Dynamic multilevel dual queue scheduling algorithms for grid computing," in *Software Engineering and Computer Systems*, vol. 179 of *Communications in Computer and Information Science*, pp. 425–440, Springer, Berlin, Germany, 2011.

[11] L. A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, no. 3, pp. 338–353, 1965.

[12] L. A. Zadeh, "Making computers think like people," *IEEE Spectrum*, vol. 8, pp. 26–32, 1983.

[13] R. E. Bellman and L. A. Zadeh, "Decision making in a fuzzy environment," *Management Science*, vol. 17, no. 4, pp. 141–164, 1970.

[14] L. A. Zadeh, "Is there a need for fuzzy logic?" *Information Sciences*, vol. 178, no. 13, pp. 2751–2779, 2008.

[15] A. A. Aburas and V. Miho, "Fuzzy logic based algorithm for uniprocessor scheduling," in *Proceedings of the International Conference on Computer and Communication Engineering (ICCCE '08)*, pp. 499–504, IEEE, May 2008.

[16] S. Raheja, R. Dhadich, and S. Rajpal, "An optimum time quantum using linguistic synthesis for round robin scheduling algorithm," *International Journal of Soft Computing*, vol. 3, no. 1, pp. 57–66, 2012.

[17] A. Rezaee, A. M. Rahmani, and S. Adabi, "A fuzzy algorithm for adaptive multilevel queue management with QoS feedback," in *Proceedings of the International Conference on High Performance Computing and Simulation (HPCS '11)*, pp. 121–127, IEEE, Istanbul, Turky, July 2011.

[18] W.-L. Gau and D. J. Buehrer, "Vague sets," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, no. 2, pp. 610–614, 1993.

[19] A. Lu and W. Ng, *Vague Sets or Intuitionistic Fuzzy Sets for Handling Vague Data: Which One Is Better?*vol. 3716 of *Lecture Notes in Computer Science*, Springer, 2005.

[20] H. Bustince and P. Burillo, "Vague sets are intuitionistic fuzzy sets," *Fuzzy Sets and Systems*, vol. 79, no. 3, pp. 403–405, 1996.

[21] S. Raheja and R. Dhadich, "Many valued logics for modeling vagueness," *International Journal of Computer Applications*, vol. 61, no. 7, pp. 35–39, 2013.

[22] S. Raheja, R. Dhadich, and S. Rajpal, "Designing of 2-stage CPU scheduler using vague logic," *Advances in Fuzzy Systems*, vol. 2014, Article ID 841976, 10 pages, 2014.

[23] R. J. Matarneh, "Self-adjustment time quantum in round Robin algorithm depending on burst time of the now running processes," *American Journal of Applied Sciences*, vol. 6, no. 10, pp. 1531–1537, 2009.

[24] M. Park, H. J. Yoo, J. Chae, and C.-K. Kim, "Quantum-based fixed priority scheduling," in *Proceedings of the International Conference on Advanced Computer Theory and Engineering (ICACTE '08)*, pp. 64–68, Phuket, Thailand, December 2008.