Egyptian Informatics Journal (2016) **17**, 125–137

Cairo University

Egyptian Informatics Journal

www.elsevier.com/locate/eij
www.sciencedirect.com

FULL-LENGTH ARTICLE

Designing of vague logic based multilevel feedback queue scheduler

Supriya Raheja [a,*], **Reena Dadhich** [b], **Smita Rajpal** [c]

[a] *Department of Computer Science & Engineering, School of Engineering & Technology, NCU (Formerly ITM University), Gurgaon, Haryana, India*
[b] *Department of Computer Science & Informatics, University of Kota, Kota, Rajasthan, India*
[c] *Alpha Global IT, Toronto, Canada*

Received 28 January 2015; revised 2 August 2015; accepted 5 September 2015
Available online 19 October 2015

KEYWORDS

Vague based Multilevel Feedback Queue (VMLFQ) Scheduler;
Scheduler;
Multilevel Feedback Queue (MLFQ) Scheduling;
Vague set theory;
Vague inference system

Abstract Multilevel feedback queue scheduler suffers from major issues of scheduling such as starvation for long tasks, fixed number of queues, and static length of time quantum in each queue. These factors directly affect the performance of the scheduler. At many times impreciseness exists in attributes of tasks which make the performance even worse. In this paper, our intent is to improve the performance by providing a solution to these issues. We design a multilevel feedback queue scheduler using a vague set which we call as VMLFQ scheduler. VMLFQ scheduler intelligently handles the impreciseness and defines the optimum number of queues as well as the optimal size of time quantum for each queue. It also resolves the problem of starvation. This paper simulates and analyzes the performance of VMLFQ scheduler with the other multilevel feedback queue techniques using MatLab.

1. Introduction

A scheduler is the key module of any contemporary operating system that manages the concurrent execution of active tasks by sharing the CPU time among these tasks. To achieve these goals it runs a scheduling algorithm which selects the next task to run as well as divide the CPU time. In a productive system, scheduler should be fair and efficient [1,2]. Efficiency and fairness can be considered in terms of different parameters such as average waiting time, average turnaround time, average response time, and starvation. These goals vary with the system being used. Keeping these goals, operating system's designers prefer to use Multilevel Feedback Queue (MLFQ) scheduling algorithm for scheduler over other scheduling algorithms.

The fundamental problems with the MLFQ scheduling are threefold: first is how to assign the parameters to the scheduler, such as how to decide the optimum number of queues, how much length of time quantum for each queue and how the priority is assigned to each task, so that starvation will not occur

* Corresponding author.
Peer review under responsibility of Faculty of Computers and Information, Cairo University.

Production and hosting by Elsevier

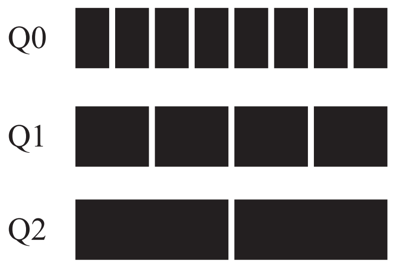

Figure 1 Variation in the size of time quantum within different queues.

[3]. Most of the MLFQ schedulers allow variable length of time quantum to the queues. Often higher priority queues are assigned short time quantum for interactive tasks and the lower priority queues are assigned long time quantum as they contain background tasks. Fig. 1 illustrates the variation in size of time quantum within each queue. Each black box represents the one time quantum.

Second, it tries to optimize the average turnaround time. Third, MLFQ desires that the system should be more responsive, thus to minimize the response time [4]. However, the algorithms such as Round Robin minimize the response time but unfortunately increase the turnaround time [5]. Moreover, task's attributes can be having imprecise data which further affect these issues and make performance even worse. Hence, the focus of an operating system designer is to build a scheduler that achieves all the desired goals of scheduling and at the same time handles the impreciseness.

In this paper we introduce a vague logic based new multilevel feedback queue CPU scheduler and call it as VMLFQ scheduler. VMLFQ scheduler considers all the above mentioned problems with MLFQ and provides solutions to all. VMLFQ scheduler dynamically calculates the length of time quantum for each queue which makes the scheduler flexible. Hence, it can take decisions at run time. With all these, it also improves the performance of a system in terms of average waiting time, average turnaround time, average normalized turnaround time and average response time.

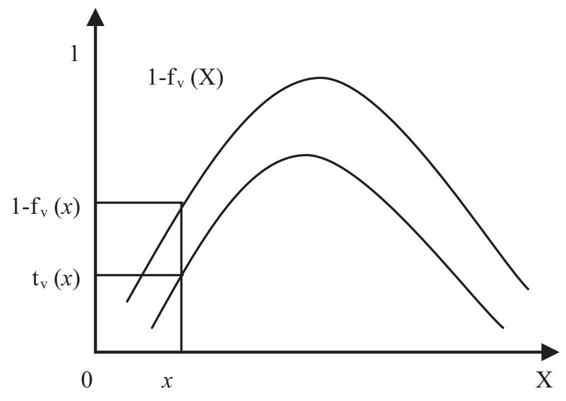

Figure 3 Vague member functions.

This paper is organized as follows. Section 2 gives brief explanation of the task scheduling algorithm and the Multilevel Feedback Queue Scheduling. This section also discusses the related work of MLFQ. Section 3 provides the reader with the background information on vague set theory. Section 4 describes VMLFQ scheduler in detail. Section 5 discusses the simulation with the help of sample task sets and results. Finally, Section 6 concludes the work.

2. Related work

Scheduling algorithm is the technique that a scheduler uses to decide the next task to run. The performance of operating system mainly depends on the scheduling algorithm used by scheduler. There have been a number of scheduling algorithms proposed in the literature such as First Come First Serve, Priority, Shortest Job First, Round Robin, Multilevel Queue, and Multilevel FeedBack Queue scheduling algorithm [6–9]. However, out of all, multitasking systems prefer Multilevel Feedback Queue Scheduling algorithm [10,11]. As our focus is on Multilevel Feedback Queue, all these algorithms are out of scope of this paper.

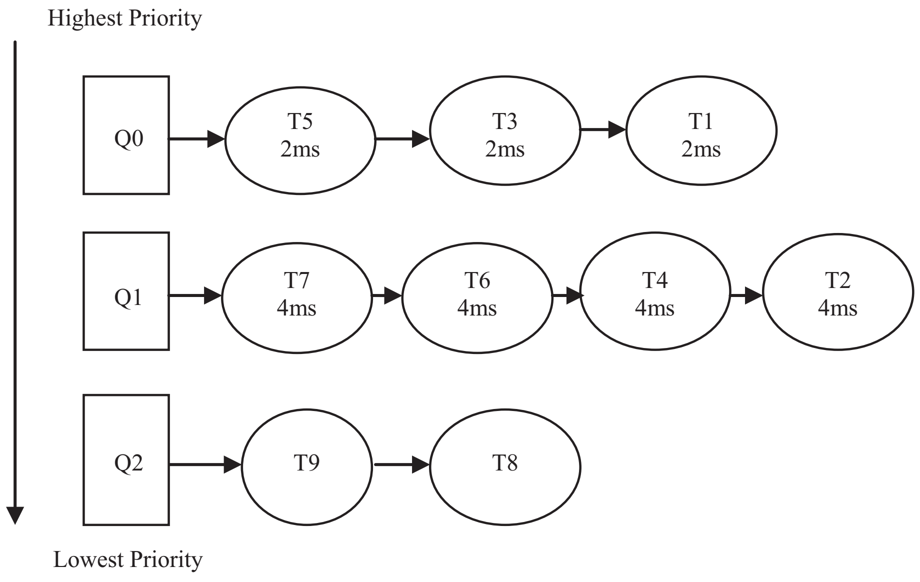

Figure 2 Pictorial representation of multilevel feedback queue example.

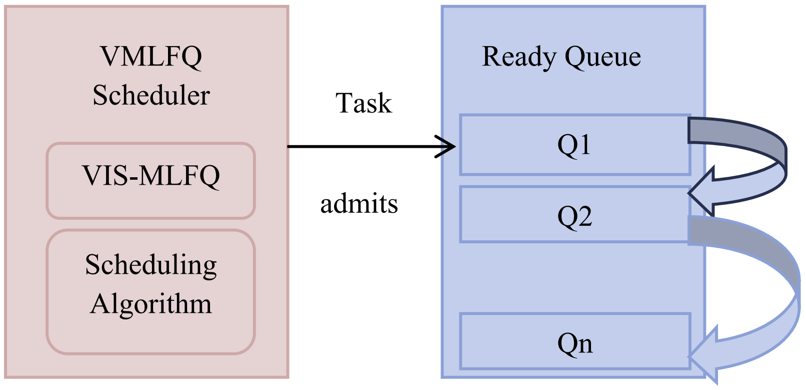

Figure 4 VMLFQ scheduler.

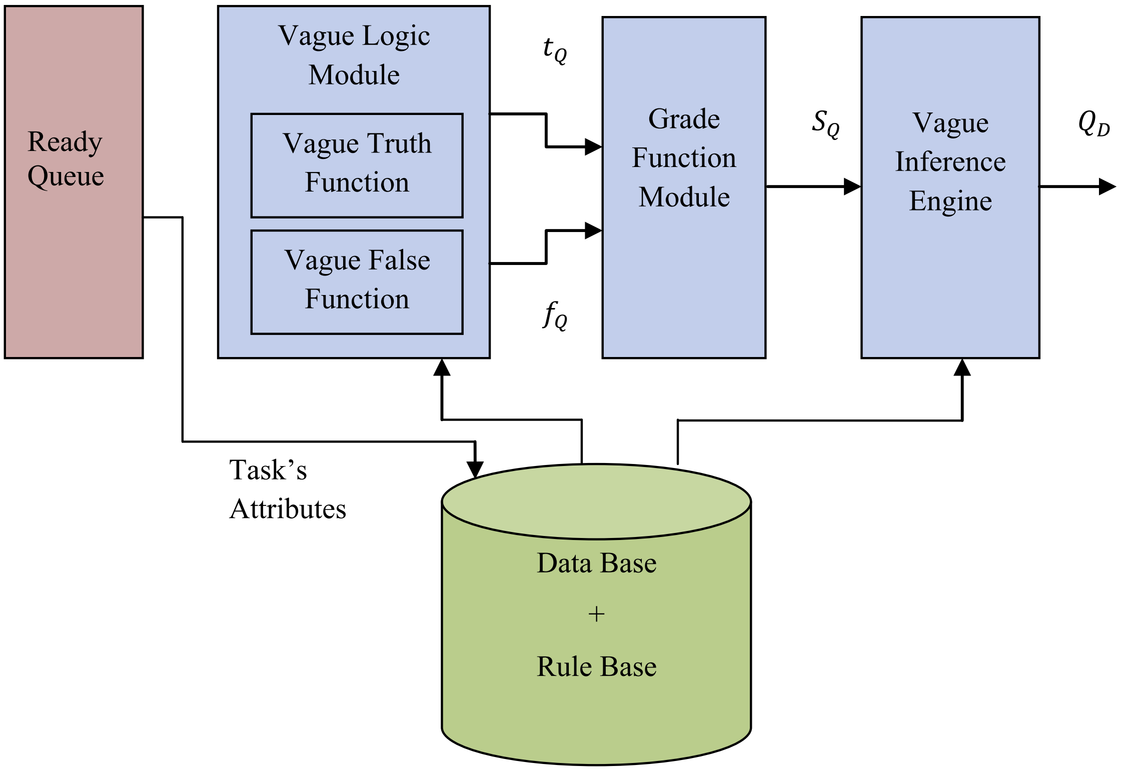

Figure 5 Vague inference system for VMLFQ scheduler.

MLFQ scheduling is one of the widely known scheduling approaches for interactive systems. In 1962, Corbato et al. had first accounted the MLFQ scheduler in a system called as Compatible Time Sharing System (CTSS). Just like Multilevel Queue, the Multilevel Feedback Queue also contains multiple queues and each queue has a different priority. The highest priority task from highest priority queue is scheduled first with CPU. Here, the priority is the key factor for task to run since MLFQ allows moving the task from one priority queue to another priority queue. Therefore, the fundamental principle of MLFQ scheduling depends on how to set priorities of each task. Instead of assigning a static priority to task, MLFQ changes the priority according to its observed behavior [4].

Suppose, if a task repeatedly requests for input device and releases the CPU iteratively, MLFQ scheduler will mark this task as high priority task; rather if a task requests the CPU for long duration, MLFQ will mark it as low priority task. Additionally, if a task waits for CPU for long duration in lower priority queue, it will move to the higher priority queue. In such a manner, MLFQ tries to memorize about a task, and hence uses the past activities of the task to predict its behavior. Let us consider an example of MLFQ where ready queue is divided into three queues Q0, Q1 and Q2 as shown in Fig 2. Here Q0 has higher priority than Q1, and Q2 has lowest priority. Each queue has its own scheduling algorithm. The scheduler follows RR scheduling approach for queues Q0 and Q1

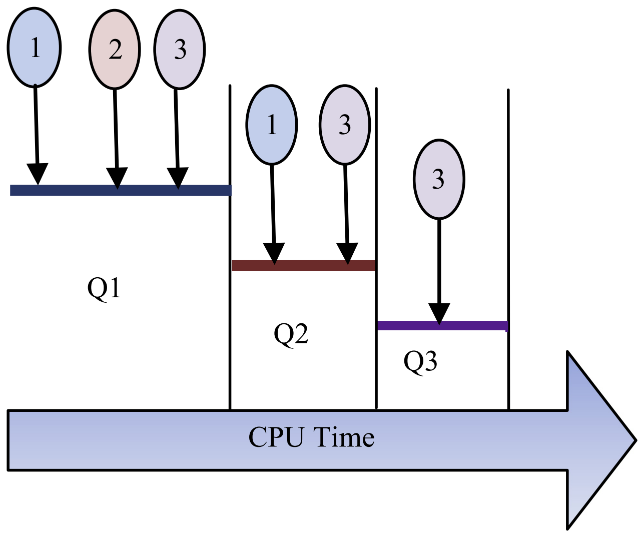

Figure 6 VMLFQ scheduling example.

Table 1 Sample task set 1.

Task name	Arrival time (ms)	Burst time (ms)
T1	0	40
T2	0	30
T3	0	50
T4	2	70
T5	4	25
T6	6	60
T7	7	45

whereas for Q2, it follows FCFS approach. When the task enters in the system, firstly it is added at the end of Q0 and then system allots a fixed single time quantum. This scheduling algorithm provides the facility to move the tasks from one queue to another queue. If the task consumes more CPU time, the task is moved to the lower priority queue Q1 and allotted double time quantum.

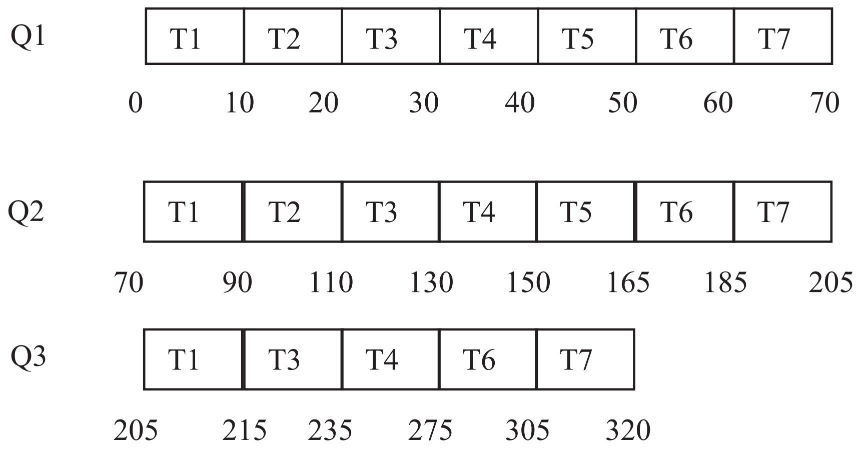

Figure 7 Gantt chart for sample task set 1 using MLFQ.

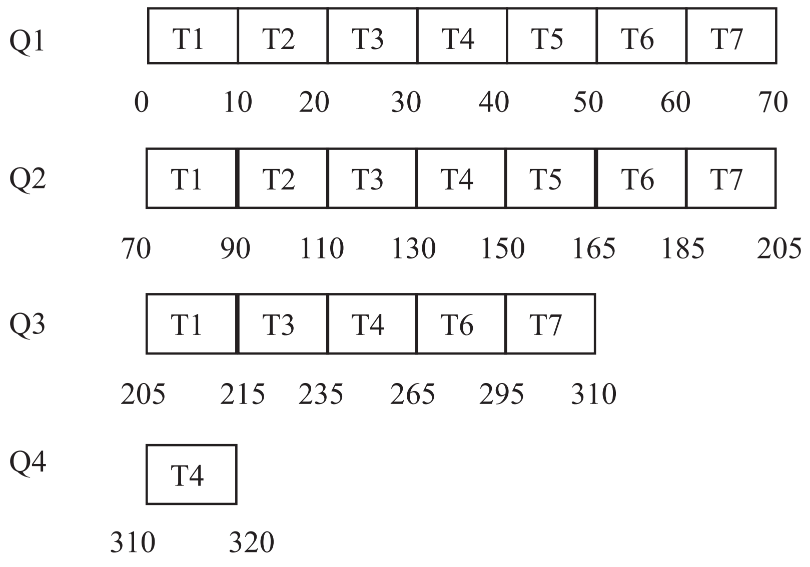

Figure 8 Gantt chart for sample task set 1 using PMLFQ.

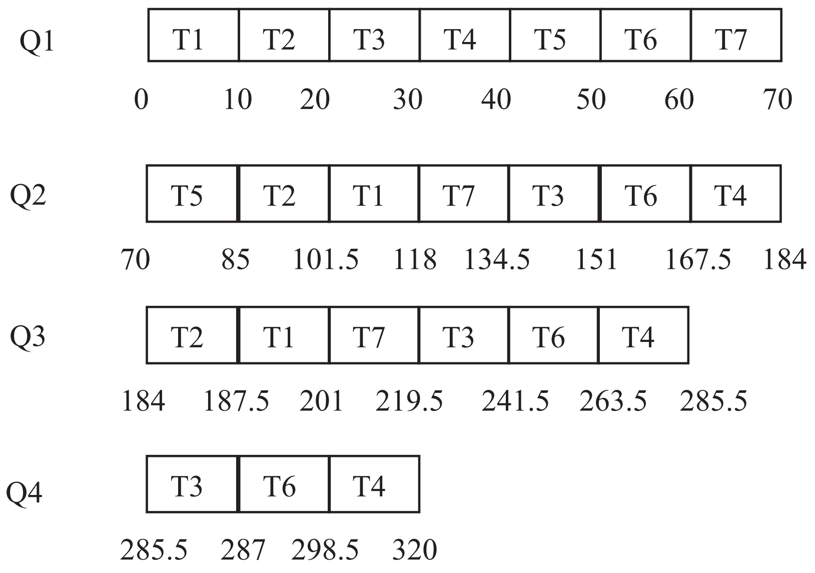

Figure 9 Gantt chart for sample task set 1 using VMLFQ.

Q0 is using time quantum 2 ms and Q1 is using 4 ms as illustrated in Fig 2. In this example, the interactive and the I/O bound tasks (T1, T3 and T5) will complete their execution in Q0 but the CPU bound tasks (T2, T4, T6, T7, T8 and T9) will move to the lower level queues Q1 and Q2. The task that waits for a longer time can be moved to the higher priority queues.

Since MLFQ is mainly preferable algorithm for interactive tasks, Torrey and Coleman have implemented a MLFQ scheduler to compare the response of interactive tasks. Proposed algorithm has shown some improvements in terms of turnaround time, as the MLFQ scheduling performance depends on the number of queues and the length of time quantum is assigned to each queue. With this aim, Parvar and Safari have utilized the recurrent neural network to optimize the number of queues and the size of time quantum of each queue of MLFQ scheduler [12]. Hoganson has pointed the performance of MLFQ scheduler in terms of task starvation. MLFQ scheduling algorithm is efficient and effective for small CPU bound tasks or interactive tasks but in the lower priority queues, the CPU-intensive tasks may get starved and reduce the performance of MLFQ scheduler. He has presented an approach that extenuates the MLFQ starvation problem [13].

Rao and Shet have further implemented the MLFQ scheduler for multi-task real time systems [14]. They have proposed a new multilevel feedback queue (NMLFQ) scheduling algorithm which is implemented in C++. Bhunia has also given a solution for the MLFQ scheduler for the tasks which get starved in the lower priority queues for waiting CPU. The proposed solution considers five queues. In this approach, the number of queues is fixed and also the time slice increases from upper to lower queues. The tasks from the lower queue may also be shifted to higher priority queues on the basis of remaining CPU burst time which results in the reduction of number of starved tasks up to some level [15]. We are calling this algorithm as PMLFQ scheduling algorithm. In this book Arpaci-Dusseau and Arpaci-Dusseau have discussed numerous issues related to MLFQ scheduling [4]. There is no development in the literature which shows the handling of uncertainty in MLFQ scheduling.

This research work is grounded to handle the uncertainty using vague set theory. We call this as VMLFQ scheduler. We compare VMLFQ with MLFQ and PMLFQ scheduling. We claim that the proposed work results in better performance in terms of average waiting time, average response time, average normalized turnaround time and average normalized turnaround time. In the next section we will discuss preliminaries of vague set theory which is the core part of our work.

3. Vague set theory

In this section we concisely discuss one of the generalized forms of fuzzy set theory called as vague set theory. Fuzzy set theory was specifically designed by Prof. Zadeh to mathematically represent impreciseness and uncertainty [16–18]. It is a formalized tool for dealing with imprecision. Let $X = \{x_1, x_2, \ldots, x_n\}$ be the universe of discourse, where an element of X is denoted by x.

Definition 1 (*Fuzzy Set*). A fuzzy set F in X is defined by its membership function μ_F: $X \rightarrow [0, 1]$ where $\mu_F(x)$ is the degree of membership of element x in a fuzzy set F [19–22].

In a fuzzy set, each element is assigned a single membership value in the interval [0,1]. This single membership value does not separate the evidence in favor and evidence against. Gau and Buehrer have pointed this single membership value and introduced vague set theory over fuzzy set theory which considers both evidences individually [23].

Definition 2 (*Vague Set*). A vague set V in X is defined by a truth membership function $t_v(x) \in [0, 1]$and a false membership function $f_v(x) \in [0, 1]$, where $t_v(x)$ is a lower bound on the grade of membership of x derived from the evidence for x, $f_v(x)$ is a lower bound on the grade of membership of x derived from the evidence against x, and $t_v(x) + f_v(x) \leqslant 1$. The grade of membership of x in the vague set is bounded to a subinterval $[t_v(x), 1 - f_v(x)]$ of [0, 1] as shown in Fig. 3 [23,24].

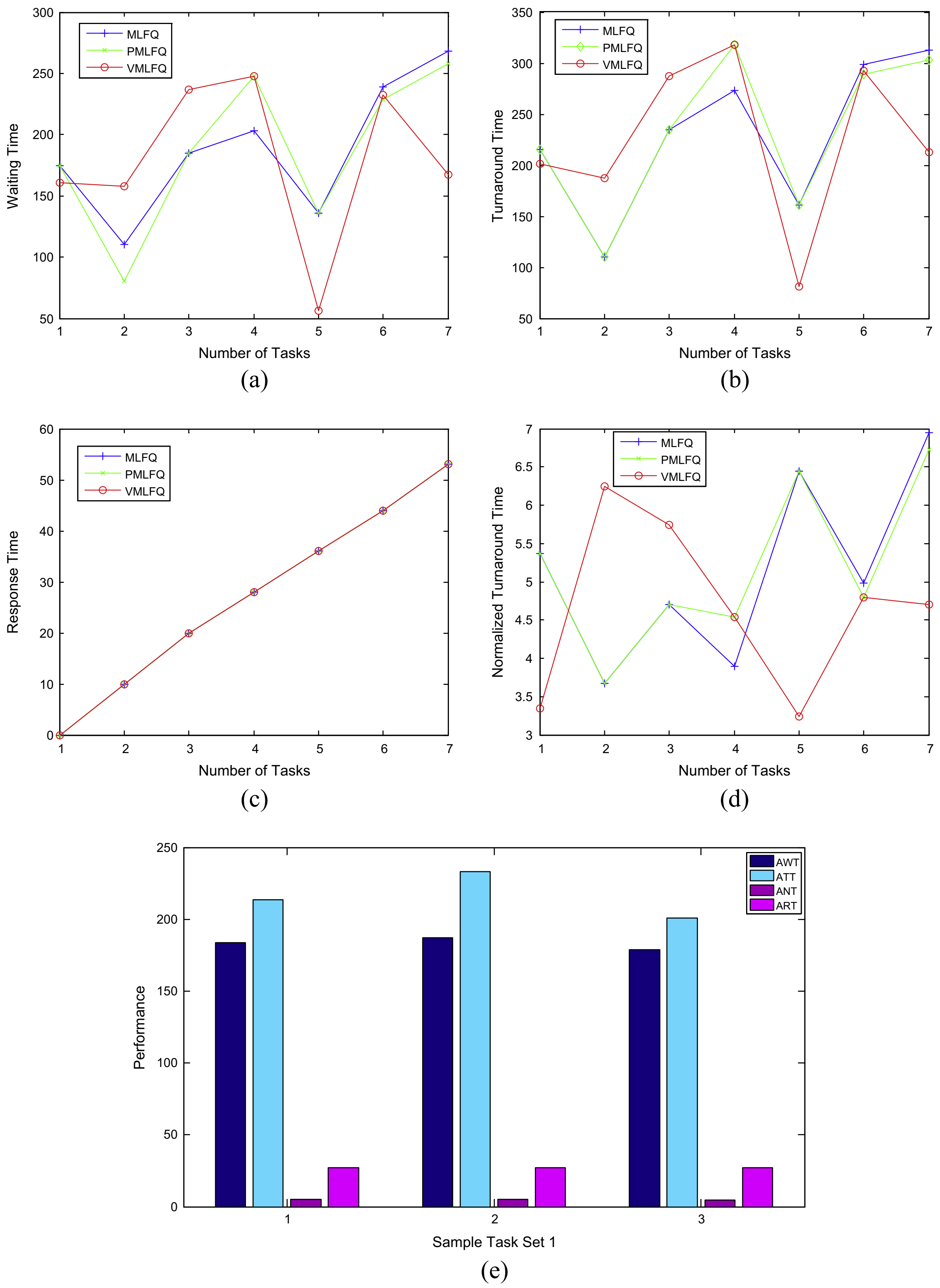

Figure 10 (a) Waiting time (task set 1), (b) turnaround time (task set 1), (c) response time (task set 1), (d) normalized turnaround time (task set 1) and (e) overall performance result (task set 1).

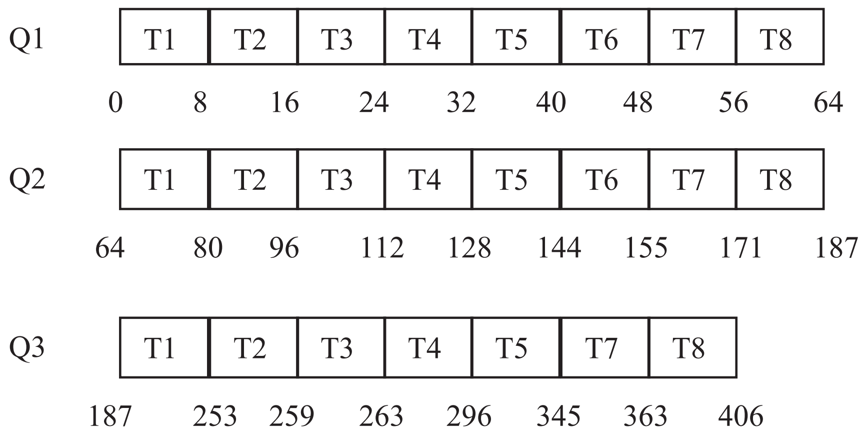

Figure 11 Gantt chart for MLFQ (sample task set 2).

Table 2 Sample task 2.

Task name	Arrival time (ms)	Burst time (ms)
T1	0	90
T2	0	30
T3	0	28
T4	0	57
T5	0	73
T6	0	19
T7	0	42
T8	0	67

Definition 3 (*Vague Value*). The interval $[t_v(x), 1 - f_v(x)]$ is called the 'vague value' of x in V as given in Fig 3. The vague set V is written as $V \leqslant x, [t_v(x), f_v(x)] > : x \in X$.

Consider a universe X for priority of the tasks. A vague value for priority of a task can be [0.5, 0.7]. Here, 0.5 and .3 represent the truth part and false part of vague value respectively, whereas remaining 0.2 stands for hesitated part of vague value [25].

4. VMLFQ scheduler

VMLFQ scheduler supports a finite number of queues n and a finite number of active tasks N. Each task requires arrival time A and burst time B. Let $Q = \{Q_1, Q_2, \ldots, Q_n\}$ be the set of n queues and $T = \{T_1, T_2, \ldots, T_N\}$ be the set of N tasks, then $\{A_i | i = 1, \ldots, N\}$ represents the arrival time and $\{B_i | i = 1, \ldots, N\}$ represents the burst time for ith task respectively. VMLFQ scheduler dynamically divides the ready queue into finite number of queues $Q_1, \ldots, Q_n \in Q$. VMLFQ scheduler has two components as shown in Fig. 4:

- Vague Inference System (VIS-MLFQ)
- Scheduling algorithm

4.1. VIS-VMLFQ

Our VIS-MLFQ has the ability to learn the current behavior of the active tasks and based on the ability, it converts the inputs

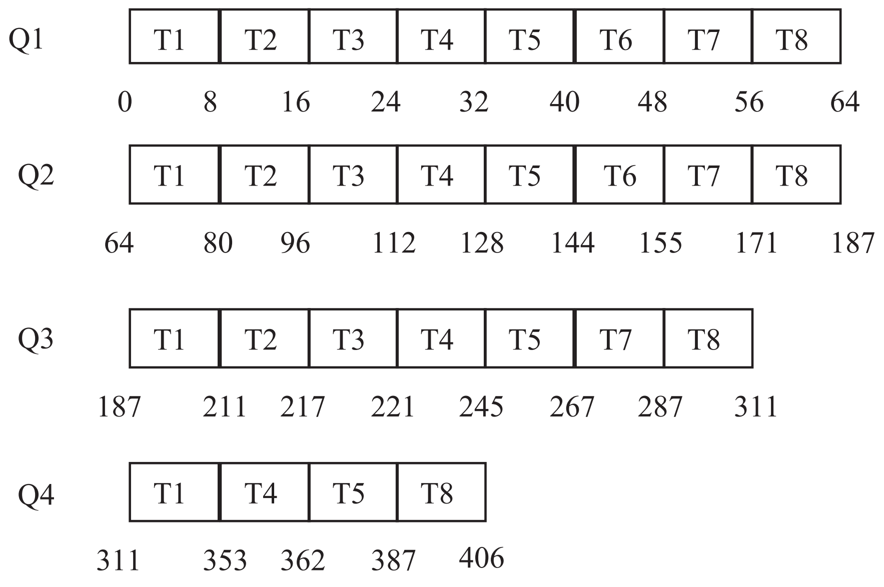

Figure 12 Gantt chart for PMLFQ (sample task set 2).

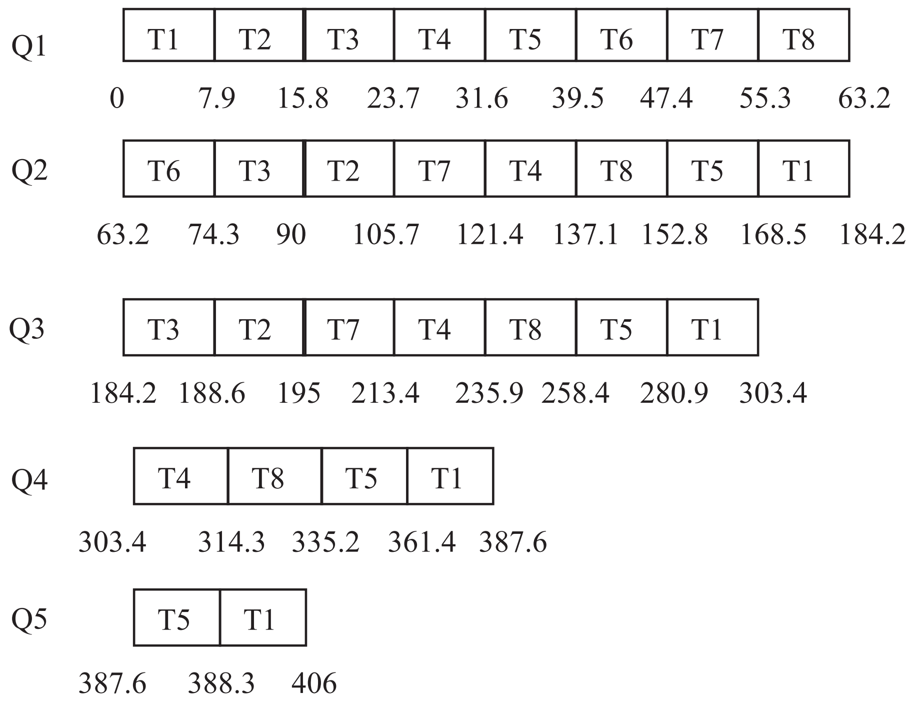

Figure 13 Gantt chart for VMLFQ (sample task set 2).

into the desired output. Here, our aim is to generate an optimum value for time quantum. Hence, our VIS takes the burst time and number of tasks as inputs and maps these inputs into an optimum size of time quantum. Fig. 5 illustrates the flow of data from one module to another module in VIS-VMLFQ. It has 4 modules as given below:

- Vague Logic Module
- Grade Function Module
- Vague Inference Engine
- Data Base and Rule Base

4.1.1. Vague Logic Module

VLM converts the inputs into the respective vague values to handle the dependability of task's parameters in the universe of discourse [0, 1]. These vague values are defined with true-membership function (t_Q) and false-membership function (f_Q). VLM takes two parameters, burst time (B) and number of active tasks (N) as inputs from the Data Base. Based on these inputs t_Q and f_Q are defined as given in Eq. (1) and Eq. (2) respectively.

$$t_Q = \frac{B_{avg}}{B_{avg} + B_{max} + N} \tag{1}$$

$$f_Q = \frac{B_{avg}}{B_{avg} + B_{min} + N} \tag{2}$$

Though t_Q and f_Q functions are independent to each other a relation is drawn between these two vague functions i.e. $t_Q + f_Q \leqslant 1$ and $t_Q \leqslant f_Q \leqslant 1$. Since the system is not aware of the actual values of parameters and scheduler does not consider the dependability of parameters to schedule the tasks. In our work, VLM handles the uncertainty and impreciseness by considering the in-between dependability of different attributes of tasks during its decision making. It results in the formation of these two membership functions t_Q and f_Q, on which decisions of our scheduler depend.

4.1.2. Grade Function Module

GFM defines the degree of accuracy of vague values. It receives the vague functions as input and by adding these two functions it returns the accuracy among vague value as given in Eq. (3). Similar to vague functions, the degree of accuracy S_Q should also be less than 1.

$$S_Q = t_Q + f_Q, \quad S_Q \leq 1 \tag{3}$$

4.1.3. Data Base and Rule Base

Data Base acts as the container for the ready tasks [26–28]. It retrieves all the necessary information about tasks from the ready queue and stores within it, for example burst time B, arrival time A, number of ready tasks N, static time quantum Q_S. It also maintains the information about the maximum and minimum burst time, average burst time currently present in ready queue using Eqs. (4)–(6).

$$B_{max} = \max\{B_1, B_2 \ldots\ldots B_N\} \tag{4}$$

$$B_{min} = \min\{B_1, B_2 \ldots\ldots B_N\} \tag{5}$$

$$B_{avg} = \sum_{i=1}^{N} \frac{B_i}{N} \tag{6}$$

It always fetches the current values from the ready queue and returns to VLM and Vague Inference Engine.

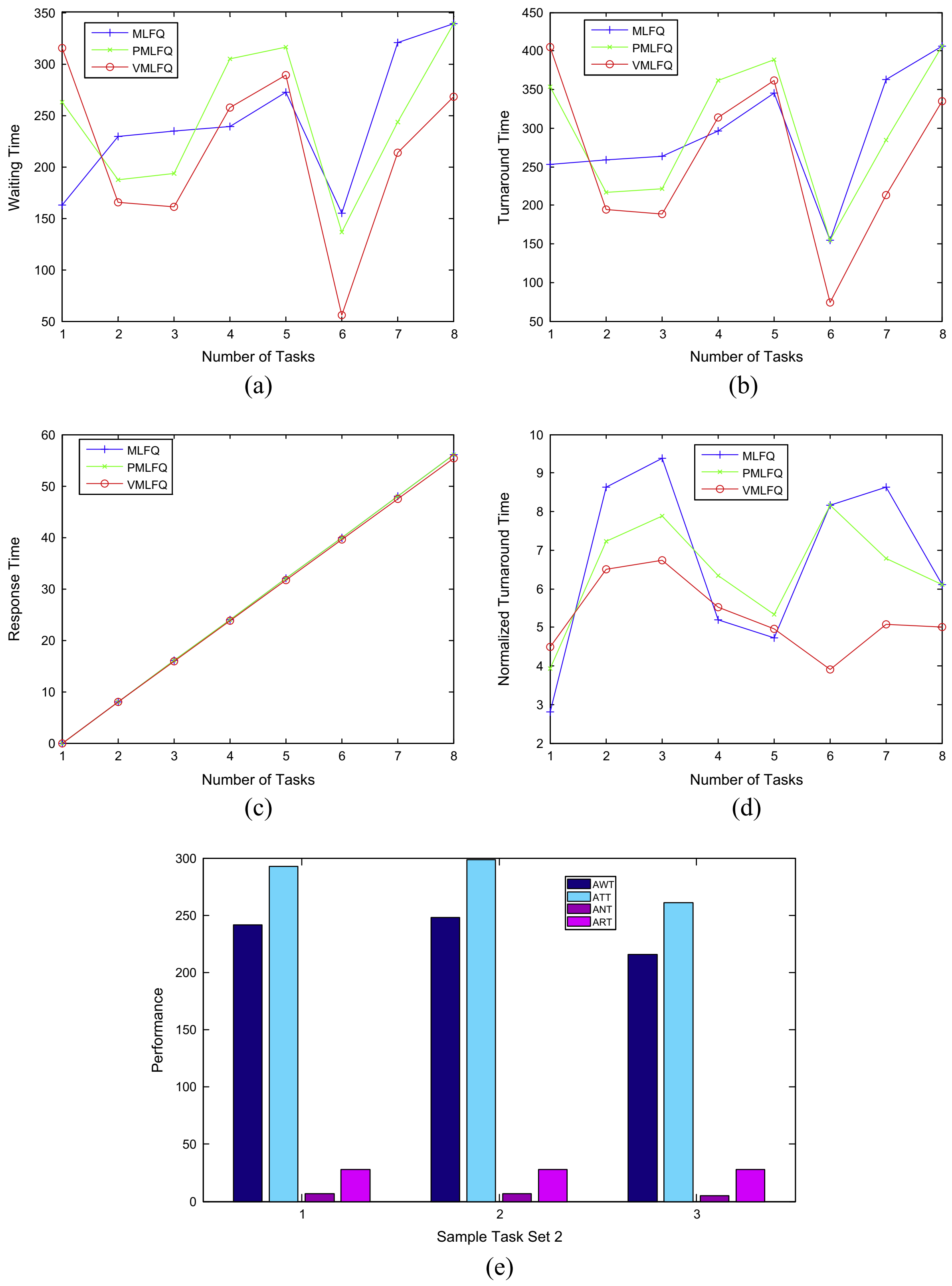

Figure 14 (a) Waiting time (task set 2), (b) turnaround time (task set 2), (c) response time (task set 2), (d) normalized turnaround time (task set 2) and (e) overall performance result (task set 2).

4.1.4. Vague Inference Engine (VIE)

VIE returns the optimum value of time quantum Q_D. It fetches the value of Q_S from the Data Base and the degree of accuracy from the GFM. Finally, on the basis of rules given in Eq. (7), it returns the size of time quantum Q_D.

if $(\forall i, A_i == 0)$ then

$$Q_D = S_Q \times Q_S \tag{7}$$

else

$$Q_D = Q_S$$

4.2. VMLFQ Algorithm

```
Begin
  Initialize the variables
    $Q_S$ = static time quantum assigned by the system
    $N$ = number of tasks
  Do Loop: 1, …, N
    Initialize the variables
      $B$ = burst time of task.
      A = arrival time of task.
  // initial value of remaining burst time.
    $RBT = B$
  End Loop
  Do Loop: 1, …, N
    Assign all tasks to Q1.
    Calculate $Q_D$ for Q1    // Using VIS-MLFQ
  End Loop
  Do Loop: 1, …, N
    Calculate the response ratio (RR) using Eq. (8).

$$RR = \frac{Waiting\ Time + Burst\ Time}{Burst\ Time} \tag{8}$$

  End Loop
  Sort the Queue in descending order of RR.
  Schedule the tasks for $Q_D$ time.
  if ($RBT < Q_D$)
    Task does not need more CPU time
  else
    Task moves to lower level queue $Q_i$ where $2 \leqslant i \leqslant n$
  End if
  Calculate the dynamic quantum for lower level queues $Q_2$ to
$Q_n$ using Eq. (9).

$$Q_D = S_Q(Q_D + N) \tag{9}$$

    Go to begin until N = = 0
End
```

4.3. Working of VMLFQ scheduler

Initially, it assigns all active tasks to highest priority queue Q1. VMLFQ scheduler sorts Q1 in the descending order of response ratio calculated using Eq. (8). It schedules the task with highest response ratio first with CPU for the length of optimum time quantum which is calculated by the component VIS-MLFQ.

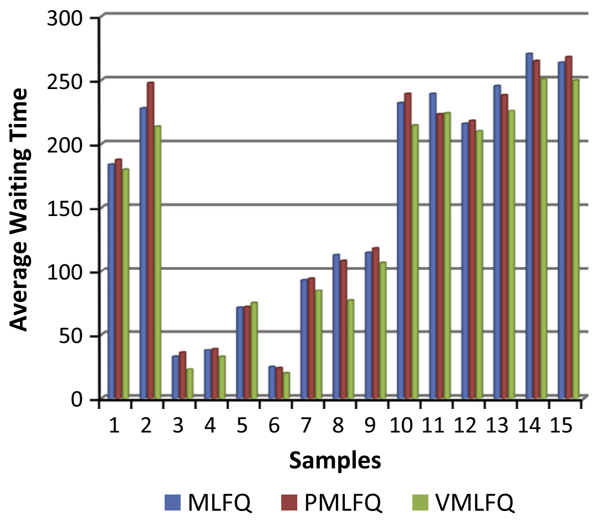

Figure 15 Average waiting time.

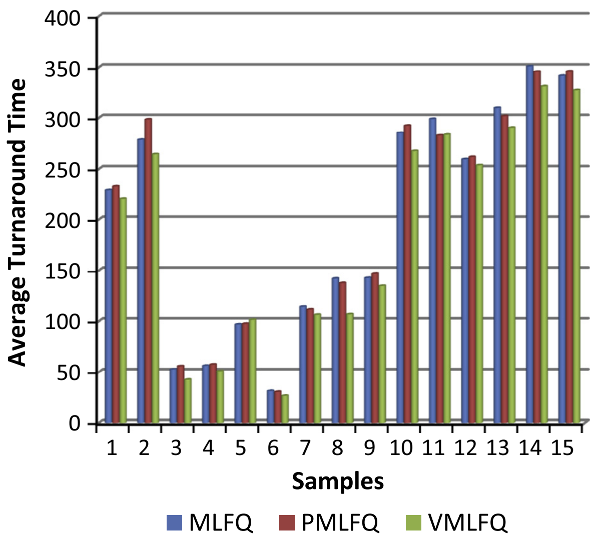

Figure 16 Average turnaround time.

The length of optimum time quantum depends on the number of tasks, their burst time, and also on the system assigned time quantum. After completing the assigned time quantum, the task will either move to next level queue Q2 or leave the system. If the task has completed its execution it will leave the system otherwise resumes its execution in the next level queue. The tasks of lower level queues will be scheduled only when the higher level queues becomes empty. As the task waits for a long time, the priority of the task will automatically move up on the basis of response ratio. Considering the response ratio for scheduling the tasks prevents the starvation problem in lower level queues [29]. The value of time quantum for other queues $Q_2, \ldots, Q_n$ depends on the value of time quantum for previous queue, number of tasks in current queue and the remaining burst time of tasks.

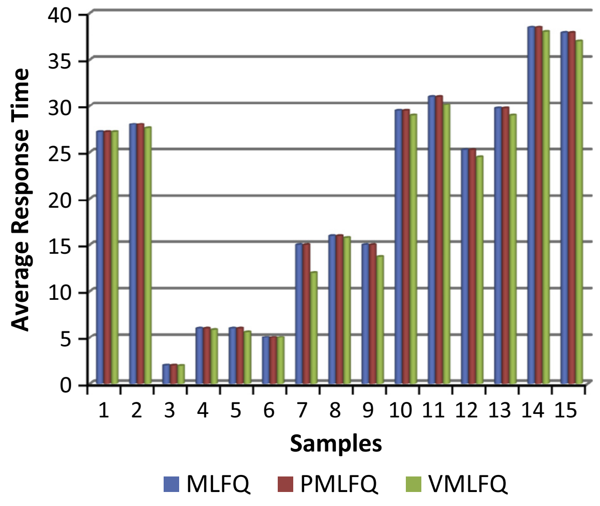

Figure 17 Average response time.

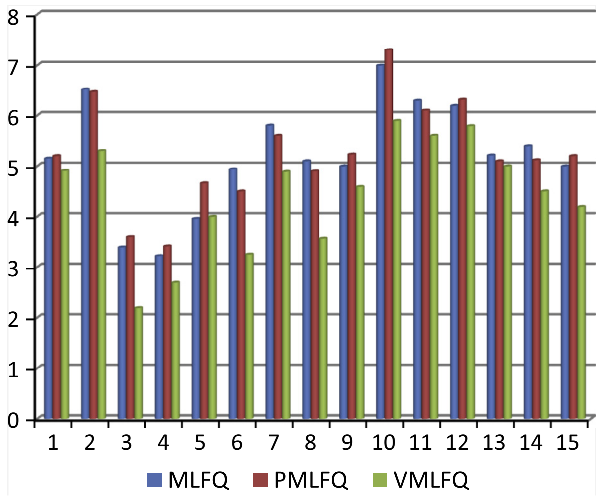

Figure 18 Average normalized turnaround time.

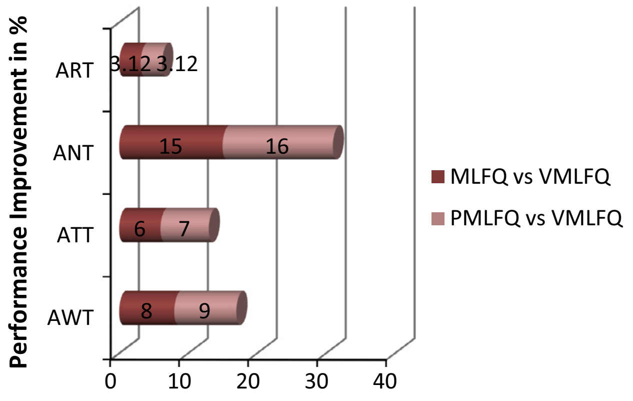

Figure 19 Improvement in performance of VMLFQ scheduler with MLFQ and PMLFQ scheduling.

Let's look at an example, where task 2 is I/O, task 1 is CPU bound and task 3 is highly CPU bound. These three tasks are initially assigned to higher priority queue Q1 (see Fig. 6). As task 2 is I/O bound, it needed less CPU time and completed its execution within the assigned time quantum (calculated through VIS) but task 1 and task 3 being CPU bound need more CPU time than the assigned time quantum. Therefore, these tasks have shifted to lower priority queue Q2. A new time quantum is assigned to the tasks in Q2. Now we can see that in Q2, task 1 has completed its execution within the time quantum but task 3 needs more CPU time so it is again shifted to Q3. In this specific example our ready queue is divided into three queues but there can be more number of queues depending on the requirement of application. Next section discusses the two components of VMLFQ scheduler in detail.

5. Simulation results

MatLab is used to design the VMLFQ scheduler. We are presenting some of the snippets for different modules defined for the VIS-MLFQ. Below is the snippet for VLM which fetches the data from the Data Base and returns the vague value.

```
function [tQ, fQ] = vague_value(b, n)
   bavg = mean(b);
   bmax = max(b);
   bmin = min(b);
      tQ = (bavg)/(bavg + bmax + n);
      fQ = (bavg)/(bavg + bmin + n);
end
```

The following snippet is for GFM that computes the grade value after extracting the membership functions tQ and fQ from VLM.

```
function [SQ] = grade_value(tQ, fQ)
   SQ = tQ + fQ;
end
```

The given snippet tells us how VIE calculates an optimum value for time quantum.

```
function[QD] = OTQ(SQ, Qs)
   if(sum = = 0)
      QD = SQ * Qs;
   else
      QD = Qs;
   end
end
```

In this section, we are considering two sample task sets to evaluate the work. These task sets show how the scheduling sequence of tasks using VMLFQ scheduler is different from other MLFQ scheduling techniques. Each sample task set is scheduled with MLFQ, PMLFQ and VMLFQ scheduling algorithms. Finally, these algorithms are compared in terms of performance metrics discussed in next section.

5.1. Performance criteria

We discuss the performance metrics which we consider for evaluating the performance of VMLFQ scheduler.

Definition 4 (*Waiting Time*). Total time a task waits in ready queue from its submission to completion. Average waiting time can be calculated as given in Eq. (10). Here $W_1, W_2, \ldots, W_N$ are the waiting times for tasks $(T_1, \ldots, T_N) \in T$ respectively and N is the total number of active tasks.

$$AWT = \frac{W_1 + W_2 + \ldots W_N}{N} \tag{10}$$

Definition 5 (*Response Time*). Time elapsed between the task submission to its first useful output. Average response time can be calculated as given in Eq. (11).

$$ART = \frac{R_1 + R_2 + \ldots R_N}{N} \tag{11}$$

Here $R_1, R_2, \ldots, R_N$ are the response times for $\forall T_i \in T$, $i = 1, \ldots, N$ respectively.

Definition 6 (*Turnaround Time*). Total time of the task from the submission to its complete execution. Turnaround time can be computed by adding the waiting time and the burst time of the task. Average turnaround time can be calculated as given in Eq. (12). Here $TT_1, TT_2, \ldots, TT_N$ are the turnaround times for tasks $(T_1, \ldots, T_N) \in T$ respectively.

$$ATT = \frac{TT_1 + TT_2 + \ldots TT_N}{N} \tag{12}$$

Definition 7 (*Normalized Turnaround Time*). It indicates the relative delay experienced by a task. It is the ratio of turnaround time to the burst time. Average normalized turnaround time can be calculated as given in Eq. (13).

$$ANT = \frac{NT_1 + NT_2 + \ldots NT_N}{N} \tag{13}$$

Here $NT_1, NT_2, \ldots, NT_N$ are the normalized turnaround times for tasks $(T_1, \ldots, T_N) \in T$ respectively.

5.2. Sample task set 1

Consider a sample task set $T = \{T_1, T_2, \ldots, T_7\}$ with different arrival times and burst times as given in Table 1. We have assumed static time quantum Q_S as 10 ms. Firstly, we have applied MLFQ scheduling algorithm over the task set T. MLFQ scheduling algorithm takes the size of time quantum for Q1 and Q2 as 10 ms and 20 ms respectively (discussed in Section 2), whereas, Q3 does not need any time quantum as it follows FCFS algorithm. The sequence of scheduling for each queue is shown in Fig. 7.

After that, we have applied the second scheduling algorithm i.e. PMLFQ. It behaves just like a MLFQ algorithm but it assumes fixed number of queues i.e. 5. Although, it has 5 queues but the size of time quantum in each queue increases in the same way as in MLFQ like 10 ms, 20 ms, 30 ms and so on. Fig. 8 shows the sequence of scheduling in each queue. As all tasks have finished their execution before reaching Q5, we have therefore represented scheduling up to Q4 only.

Finally, we have applied the VMLFQ scheduler over the same task set T. VMLFQ scheduler intelligently divides the ready queue into four sub queues $(Q_1, \ldots, Q_4) \in Q$ at run time. Initially, all tasks are assigned to Q1 as discussed in Section 4. The optimum size of time quantum Q_D for each queue is calculated using VMLFQ scheduling algorithm and it returns the size for Q1, Q2, Q3 and Q4 as 10 ms, 16.5 ms, 22 ms and 30 ms respectively. The scheduling sequence for VMLFQ scheduler is shown in Fig. 9. Based on the above Gantt charts, waiting time, turnaround time, response time and normalized turnaround time are computed using Eq. (5), Eq. (6), Eq. (7) and Eq. (8) respectively. The results are illustrated in Fig. 10(a), (b), (c), (d) respectively.

Fig. 10(e) illustrates the overall performance of all three scheduling algorithms. Here, "1" represents MLFQ scheduling, "2" represents PMLFQ scheduling and "3" represents VMLFQ scheduling algorithm. From the above graph we can perceive the reduction in the performance metrics with the VMLFQ scheduling. Reductions itself show the improvement in the performance by VMLFQ.

5.3. Sample task set 2

Let us suppose another Sample task set 2 with eight tasks $(T_1, \ldots, T_8) \in T$ with different burst times but same arrival times as given in Table 2. Now, consider the static time quantum Q_S as 8 ms. Again apply all three algorithms one by one over task set 2 to analyze the performances in more detail. Firstly, apply the MLFQ scheduling algorithm. The assigned static time quantum Q_S for first queue is 8 ms. For second and third queues the value of time quantum used was 16 ms and 24 ms respectively. Fig. 11 represents the scheduling sequence using Gantt chart for MLFQ scheduling.

Then, we have applied PMLFQ scheduling over the same task set. The time quantum used by PMLFQ scheduling for each queue is 8 ms, 16 ms, 24 ms and 32 ms respectively. The scheduling sequence for PMLQ is shown in Fig. 12.

Finally, we have applied the proposed VMLFQ approach over the task set 2. In this case it divides the ready queue into five sub queues $(Q_1, \ldots, Q_5) \in Q$. The VIS returns the value of Q_D as 7.9 ms i.e. used by scheduler as time quantum for Q1 whereas time quantum for Q2, Q3, Q4 and Q5 is 15.7 ms, 22.5 ms, 26.2 ms and 29 ms respectively. The scheduling sequence for VMLFQ is given in Fig. 13.

Similarly for task set 2, waiting time, turnaround time, response time and normalized turnaround time are computed for each task and illustrated in Fig. 14(a), (b), (c) and (d) respectively.

Average waiting time AWT, average response time ART, average turnaround time ATT and average normalized turnaround time ANT are computed using Eqs. (5), (6), (7) and (8) respectively and are shown in Fig. 14(e). From the graph shown in Fig. 14(e), we can verify the improvement in the overall performance of VMLFQ as compared to MLFQ and PMLFQ. Randomly multiple sets of tasks were generated and simulated using MatLab. All task sets were scheduled using these three algorithms. The performance comparison of these algorithms is illustrated in Fig. 15, Fig. 16, Fig. 17 and Fig. 18 for average waiting time, average response time, average turnaround time, and average normalized turnaround time respectively.

In the above graphs, red line represents the VMLFQ scheduling algorithm and in each case the outputs of VMLFQ algorithm is on lower side of the graph which shows the reduction in the values of performance metrics. The reduction itself proves the improvement in the performance of scheduler as well as in the performance of system.

VMLFQ scheduler performs 3.12% better in average response time, 15% in average normalized turnaround time, 6% in average turnaround time and 8% in average waiting time as compared to MLFQ scheduling whereas it performs 3.12% better in average response time, 16% in average normalized turnaround time, 7% in average turnaround time and 9% in average waiting time as compared to PMLFQ scheduling as shown in Fig. 19.

VMLFQ scheduler performs better mainly for four reasons: The proposed VMLFQ scheduler responds effectively to dynamic environment where number of queues are assigned at run time as well as time quantum to each queue is also provided at run time. It handles the uncertainty and impreciseness of tasks. In addition, VMLFQ scheduler improves the starvation problem at the lower priority queues as we are considering the response ratio. Last but not the least, it improves the performance of the system in terms of average waiting time, average response time, average turnaround time and average normalized turnaround time.

6. Conclusions

The purpose of our work is to present a novel approach to multilevel feedback queue CPU scheduling. This paper discussed the problems and issues associated with the multilevel feedback queue (MLFQ) scheduling. To resolve the issues we presented a MLFQ scheduler that follows a vague logic approach to take the decisions while meeting the performance requirements. VMLFQ scheduling extends the concept of MLFQ scheduling using vague logic to consider the impreciseness associated with the scheduling parameters. The scheduler takes decisions based on the burst time and the number of tasks. The VMLFQ based scheduler provides the solution to each of the problems and issues. It has resolved the problem of starvation for lower level tasks. VMLFQ scheduler also followed the dynamic approach for assigning number of queues rather than fixed number of queues. Additionally, we compared the performance of VMLFQ scheduling algorithm with the MLFQ scheduling algorithm and PMLFQ scheduling algorithm via average response time, average waiting time, average turnaround time and average normalized turnaround time. Based on the results shown in Section 5, we concluded that the VMLFQ scheduling algorithm has better performance over the traditional MLFQ scheduling and PMLFQ scheduling algorithms.

References

[1] Tanenbaum AS, Woodhull AS. Operating systems design and implementation. 3rd ed.; 2006.

[2] Silberchatz A, Galvin PB, Gagne G. Operating systems concepts. 8th ed. John Wiley and Sons; 2012.

[3] Torrey LA, Coleman J, Miller BP. A comparison of interactivity in the Linux 2.6 scheduler and an MLFQ scheduler. Softw Pract Experience 2007;37(4):347–64.

[4] Arpaci-Dusseau RH, Arpaci-Dusseau AC. Operating systems: three easy pieces, scheduling: multilevel feedback queue. Version 0.80; 2014.

[5] Raheja S, Dhadich R, Rajpal S. An optimum time quantum using linguistic synthesis for round Robin CPU scheduling algorithm. Int J Soft Comput 2012;3(1):57–66.

[6] Nie B, Du J, Xu G, Liu H, Yu R, Wen Q. A new operating system scheduling algorithm. Advanced research on electronic commerce, web application, and communication, vol. 143. Springer; 2011. p. 92–6.

[7] Dhamdhere DM. Operating systems a concept based approach. 2nd ed. Tata McGraw-Hill; 2008.

[8] Milenkovic M. Operating system concepts and design. International Edition. McGraw Hill; 1992.

[9] Haldar S, Aravind AA. Operating systems. Pearson Education India; 2009.

[10] Abawajy JH. Job scheduling policy for high throughput computing environments. In: Ninth IEEE international conferences on parallel and distributed systems, Ottawa, Ontario, Canada; 2002.

[11] Feitelson DG, Rudolph L, Schwiegelshohn U. Parallel job scheduling, a status report. In: 10th international conference on job scheduling strategies for parallel processing, Springer-Verlag; 2004. p. 1–16.

[12] Parvar MRE, Parvar ME, Safari S. A starvation free IMLFQ scheduling algorithm based on neural network. Int J Comput Intell Res 2008;4(1):27–36.

[13] Hoganson KE. Reducing MLFQ scheduling starvation with feedback and exponential averaging. Consort Comput Sci Coll 2009;25:196–202.

[14] Rao MV, Shet KC. Analysis of new multi-level feedback queue scheduler for real time kernel. Int J Comput Cognit 2010;8(3):5–16.

[15] Bhunia A. Enhancing the performance of feedback scheduling. Int J Comput Appl 2012;18(4):11–6.

[16] Klir GJ, St Clair UH, Yuan B. Fuzzy set theory: foundations and applications. Englewood Cliffs, NJ: Prentice Hall; 1997, ISBN: 0-13-341058-7.

[17] Zimmerman J. Fuzzy set theory and its applications. Norwell, Massachusetts, U.S.A.: Kluwer Academic Publishers; 2001.

[18] Atanassov K. Intuitionistic fuzzy sets. Fuzzy Sets Syst 1998;20:87–96.

[19] Atanassov K. Intuitionistic fuzzy sets: theory and applications. New York: Physica-Verlag, Springer; 2000.

[20] Zadeh LA. Fuzzy sets. Inform Control 1965;8:338–56.

[21] Zadeh LA. The role of fuzzy logic in the management of uncertainty in expert systems. Fuzzy Sets Syst 1983;11:197–227.

[22] Zadeh LA. Outline of a new approach to the analysis of complex systems and decision processes. IEEE Trans Syst, Man Cybern 1973;3:28–44.

[23] Gau WL, Buehrer DJ. Vague sets. IEEE Trans Syst, Man Cybern 1993;23:610–4.

[24] Bustince H, Burillo P. Vague sets are intuitionistic fuzzy set. Fuzzy Sets Syst 1996;79:403–5.

[25] Lu A, Ng W. Vague sets or intuitionistic fuzzy sets for handling vague data: which one is better? Lecture notes in computer science, vol. 3716. Springer; 2005. p. 401–16, Conceptual Modeling-ER2005.

[26] Raheja S, Dadhich R, Rajpal S. Many valued logics for modeling vagueness. Int J Comput Appl 2013;61(7):35–9.

[27] Raheja S, Dadhich R, Rajpal S. Designing of 2-stage CPU scheduler using vague logic. Adv Fuzzy Syst 2014:1–10.

[28] Raheja S, Dadhich R, Rajpal S. 2-Layered architecture of vague logic based multilevel queue scheduler. Appl Comput Intell Soft Comput 2014:1–12.

[29] Moallemi A, Asgharilarimi M. A fuzzy scheduling algorithm based on highest response ratio next algorithm. Computer sciences and software engineering. Springer; 2008. p. 75–80.