World Academy of Science, Engineering and Technology
International Journal of Computer and Systems Engineering
Vol:2, No:9, 2008

IMLFQ Scheduling Algorithm with Combinational Fault Tolerant Method

MohammadReza EffatParvar, Akbar Bemana, and Mehdi EffatParvar

***Abstract*—**Scheduling algorithms are used in operating systems to optimize the usage of processors. One of the most efficient algorithms for scheduling is Multi-Layer Feedback Queue (MLFQ) algorithm which uses several queues with different quanta. The most important weakness of this method is the inability to define the optimized the number of the queues and quantum of each queue. This weakness has been improved in IMLFQ scheduling algorithm. Number of the queues and quantum of each queue affect the response time directly. In this paper, we review the IMLFQ algorithm for solving these problems and minimizing the response time. In this algorithm Recurrent Neural Network has been utilized to find both the number of queues and the optimized quantum of each queue. Also in order to prevent any probable faults in processes' response time computation, a new fault tolerant approach has been presented. In this approach we use combinational software redundancy to prevent the any probable faults. The experimental results show that using the IMLFQ algorithm results in better response time in comparison with other scheduling algorithms also by using fault tolerant mechanism we improve IMLFQ performance.

***Keywords*—**IMLFQ, Fault Tolerant, Scheduling, Queue, Recurrent Neural Network.

I. Introduction

IN a multi-task system, several processes are kept in the main memory and processor is kept active to run a process while the others are waiting. The key to Multi-Programming is scheduling. In the MLFQ scheduling, the processes can be dynamically moved in different queues. So processes that need a large amount of CPU time are sent to the low priority queues and process requiring I/O bound or related to interactive process are sent to high priority queues. The MLFQ scheduling organizes the queues to minimize the queuing delay and optimize the queuing environment efficiency.

In this paper, the combinational fault tolerant scheduling is presented, which analyzes the existing processes at the main memory to be executed by the CPU, and performs the time allocation in such a way that some systematic behavior is optimized. Here, the scheduler uses the previous behaviors of processes for suitable prediction of priorities to overcome the prediction problem of SJF. The scheduler evaluates the forthcoming IO activity of the processes based on the previous IO activity, so a change in process behavior results in a change of forthcoming decisions. The main contribution of this paper is to optimize the response time of the IMLFQ scheduling method by using fault tolerant mechanism, and also it tolerates the probable faults due to the consideration of the processes with the various servicing time.

The rest of the paper is organized as follows. In Section 2, a survey on scheduling methods is presented. In sections 3 IMLFQ scheduling algorithm is described respectively. Section 4 describes the combinational fault tolerant design. Section 5 gives some experimental results. Finally Section 6 concludes this work.

This work was supported in part by the Hashtroud Islamic Azad University.

MohammadReza EffatParvar is with the Young Researchers Club. Qazvin Islamic Azad University, Qazvin, Iran (e-mail: m_r_e_p@yahoo.com).

Akbar Bemana is with the Hashtroud Islamic Azad University, Hashtroud, Azarbayejan Sharghi, Iran (e-mail: a_bemana@iust.ac.ir).

Mehdi EffatParvar is with the Ardebil Islamic Azad University, Ardebil, Iran (e-mail: mehdi_effatparvar@yahoo.com).

II. Survey on Scheduling Methods

There are several scheduling algorithms which assigns processor to execute processors. There is no scheduling algorithm that works perfectly in all cases, so for a specific application we should consider several parameters such as waiting time, total response time and utilization, in algorithm selection. For example non-preemptive algorithms like FCFS and SJF are suitable when a high throughput system is needed as in batch-processing systems, and preemptive scheduling like MLFQ and Round Robin (RR) are used to provide response time and fair dispatching of CPU time as in interactive systems. The simplest scheduling algorithm that is used in most of operating systems is FCFS, which is non-preemptive minimum overhead algorithm. On the other hand, response time is not favored and no emphasis is put on throughput, damaging short and IO processes. The main advantage of this method is that no process starved. This algorithm is used in several operating systems because of its simple implementation and low overhead. FCFS is an unfair algorithm and results in weak average waiting time, while SRT and HRRN provide good response time and high overhead. RR is a fair algorithm with weak average waiting time. Moreover, SJF is an unfair algorithm with the minimum average waiting time and needs prediction. The SRT algorithm damages long processes and is liable to starvation, but because of its prediction, it has better response time in comparison with other algorithms. It is not always possible to predict the execution time of processes and there is a possibility of failure in prediction, so SRT is used theoretically.

In RR the overhead is low and there is no starvation, and this leads to a proper the response time. In this algorithm, the time slice should be selected carefully in such a way that

algorithm presents an objective behavior to have suitable overhead. Feedback scheduling algorithm works better than feedback queues in decision making and preemption in a time period schedules the processes, and consequently MLFQ is an approximation of SJF. This algorithm makes the I/O bound processes better without emphasizing on throughput, response time and possibility of starvation. In this approach, the number of queues and the time quantum are chosen by default value. MLFQ is used in interactive and I/O bound systems, the time slice between the queues is generally %80 for foreground and %20 for background. The general scheduler in Unix-based systems is based on MLFQ and some modern operating systems use MLFQ as well [9]. By taking a small quantum for layers, the response time of interactive processes is optimized; on the other hand by taking a larger quantum the throughput of the system is increased.

Generally in MLFQ scheduling different queues with different priority are used. Each queue has its own scheduling algorithm. All processes are selected from the high priority queues to execute. This method may cause starvation, and generally the low priority queues should have a higher quantum. Because of using queues, this algorithm can be easily implemented to perform the operating systems scheduling. Since this algorithm is used in many cases, its response time should be optimized in comparison with other algorithms.

Some of the problems with MLFQ are the number of priority levels of queues, finding a suitable scheduling algorithm for each queue, finding a suitable scheduling mechanism for each queue, assigning time quantum for each queue, assigning initial static priorities, adjusting dynamic priorities, favoring I/O bound processes, differentiating foreground processes and background processes, and considering client against server environment [9]. The MLFQ approach is used in IMLFQ scheduling system in such a way that the response time is decreased and the functionality of the system is improved. The optimum number of queue and the quantum for each queue are found using a fault tolerant mechanism to achieve these goals. As the proposed mechanism considers these objectives simultaneously, they do not have any negative impacts on each others. In IMLFQ scheduling, the operating system can modify the number of queues and the quantum of each queue according to the existing processes.

III. IMLFQ Scheduling Algorithm

As it was mentioned before, in MLFQ the operating system builds several separate queues and specifies the quantum for each queue. Generally in this method, all processes end in the mentioned queue and move out of system. In these methods, the number of queue and the quantum size are specified while the process is running, so the operating system has no role in controlling the number of queues and amount of each layer's quantum.

In IMLFQ, we start with indefinite numbers of queues initially. An initial value of quantum is used for each queue. When a queue is being analyzed, its quantum value is defined by $I*q$, where q is the initial value of quantum and I is the number of queues being considered [1]. For defining the numbers of layer and quantum of each layer is described in [1]. When the number of required queues and the average response time are specified based on the initial quantum of each layer, the quantum of queues should be modified in such way that the average response time of the processes are minimized [7].

According to the changes in the quantum of each queue, the movement of the processes to the lower queues is changed. So the processing time of the processes in lower layers is changed and as a result the quantum of lower layers affects the average response time. The optimized quantum has not been defined for lower queues and the average response time is related to the functionality of the whole system. Consequently the relation of the average response time and the quantum of a specified queue are not easily formulated.

To find the effect of the quantum changes, a queue should be selected and its quantum has been changed in such a way that the minimum number of processes has been moved to the lower queues [1].

Now, suppose that we have n queues in the default mode. We begin to change the quantum of layer n, since when the last queue is selected and its quantum is increased, there is no other queue to be eliminated. If the quantum of this queue is reduced a queue will be added. In this case we repeat this procedure for the newly added layer. After updating the quantum of the last queue, we continue with the previous queue, i.e. *n-1*, and change the quantum of it. The optimized average response time is specified by changing the queue *n-1*.

In this step, since there is a queue that is lower than the queue is being studied, and due to the changes made on the processes of the last queue after updating *n-1*, the optimized quantum of the last queue should be redefine. Generally, when the optimized quantum of each layer is found, the quantum of lower levels should be updated. Finally, the best average response time can be calculated using the optimized quantum of each layer.

In this step a queue is selected and its quantum is changed using RNN in a way that reserves the optimization. RNN gives the most probable model to recognize the trend information of time series data [2]. The network produces a trace of its behavior and keeps a memory of its previous states [5]. The inputs of the RNN are the quantum of queues and the average response time. Average response time, is fed to the neural network as an input, so the network finds the relation of the change of a quantum of a specified queue with the average response time and the quantum of other queues [6].

A change in the quantum of a specified queue is assumed, and tries to optimize the average response time. The neural network can find the quantum of a specified queue using the optimized quantum of lower queues. So that network finds both the quantum of the queues and the relation of increase or decrease of quantum of a queue with average response time

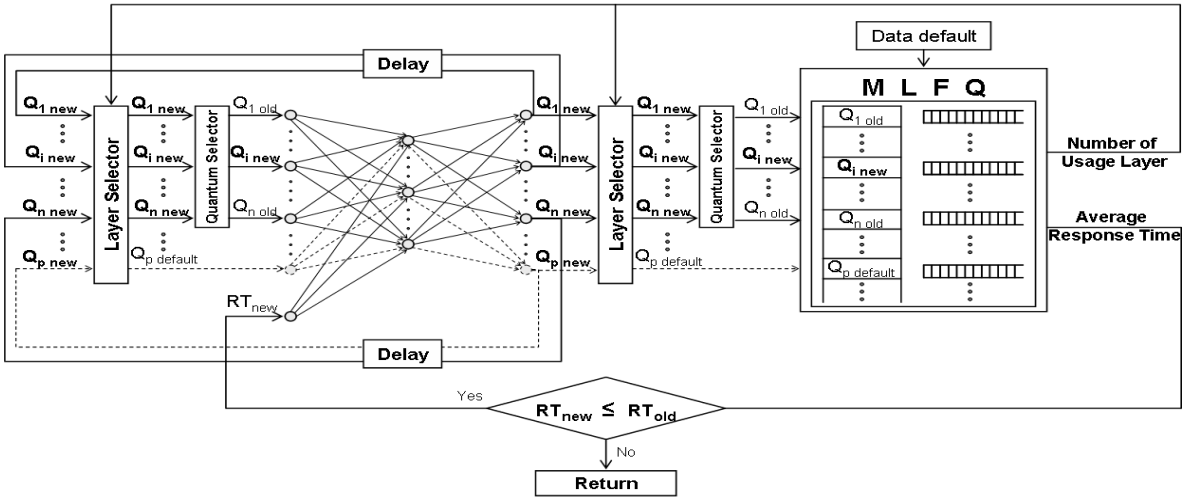

Fig. 1 Defining optimized quantum for the queue by IMLFQ function

and tries to reduce the average response time. Network updates the weights and then changes the quantum of input of queues and specifies a new quantum for queues. To find the effects of this change on average response time, the new amounts of quantum should be given to the IMLFQ function [4]. The pre-assumed processes are fed to this function and the average response time is found. Since in the previous stages, the optimized quantum for lower queues is found, it is possible that they can not be optimized any more. This situation can be prevented, when we want to replace the previous quantum with the new one, we replace the new amount of the specified queue with the former amounts. After replacing the new quantum of a specified queue in IMLFQ function, using pre-assumed default processes used to obtain the primary response time, the new average response time caused by this change is found. In this stage, when a change is applied in the quantum of a specified queue, the number of queues can be changed. Since it is possible that reducing the quantum caused more processes are moved to the lower queues or a new queue is added to the number of required queues. On the other hand increasing the quantum of a queue may cause no process is moved to the lower queues and as a result the lower queues are eliminated. The effect of the elimination and addition of a queue should be considered in the network and the new quantum for a specified queue is recognized. The MLFQ outputs are used to calculate the average response time. To equalize the entrance arrival time of these inputs with entrance time of average response time a delay function is used.

Fig. 1 shows a schematic view of the function to find the optimized quantum of the queue ***I***, and the way in which the quantum is fed in RNN and also how to limit the number of queues. When the new average response time is found, it is compared with the former one. If it is less than the previous one, the new value is selected as the input of next stage of the network to optimize the average response time. If the new value of the average response time is grated than the previous one, it means that the optimized average response time has been found.

It should be guaranteed that the learning phase is finished, and the calculated quantum is selected as the quantum of the specified queue. If the quantum of the other queues is changed, we should find their optimized quantum again. The pseudo code of the algorithm has been shown below.

IMLFQ algorithm:

1- Produce arrival time and service time for *n* process randomly using distribution function.

2- Get average response time, waiting time and maximum required layer in first stage and set the power quantum for each layer.

3- For each layer (*i=n* down to 1) update the value of queue quantum according to the maximum number of layers and average response time.

3.1- Find the optimum value of queue using RNN according to other queue quantum and the average response time that is found in the previous stages.

3.2- For each layer (*j=i+1* to *n*) repeat the step 3.2, consider the changes in other queue and update the quantum.

IV. Combinational Fault Tolerant Method

In typical applications that use operating system, fault detection and fault tolerant techniques can be implemented in software. There is a substantial redundancy in software part whereas the hardware redundancy is minimal. Due to the critical nature of the tasks in a hard real-time system, it is essential that every admitted task should be completed even in the presence of faults [9]. Therefore, fault-tolerance is an important issue in real-time systems. After designing IMLFQ algorithm we want to create a mechanism to improve this

method during the execution. We've found that such information can help developers to quickly narrow down the causes of failure [8].

In this system, we mean faults a bad response time in comparison with the other scheduling algorithm. When there is an unpredicted process in the system, a fault may appear. So we designed a fault tolerant mechanism to improve the processes response time. In almost fault tolerant mechanisms, the redundancy is needed. Since scheduling techniques are usually implemented in software, so to provide fault tolerance we should use software redundancy mechanism [10].

We used software replication method to ease IMLFQ training, to decrease the overhead and to increase the performance. Here we can not use software fault term clearly, because when a fault occurs in the IMLFQ the response time will be increased and the algorithm will be continued. In this case to stop the propagation of the faults to other processes, the neural network should be trained using suitable data. The simulation results show that when there are various processes' types with various service times, IMLFQ may not have suitable response time compared with the other scheduling algorithm. So a mechanism is needed for fault detection and the network learning.

Suppose that IMLFQ is used as the task scheduling algorithm of an operating system. If the scheduled processes which are scheduled by the system are similar to the ones that IMLFQ has been learned by, the response time of IMLFQ is better than other scheduling algorithm. But in practice, because of wide range in type of processes, IMLFQ doses not necessarily have better response time. So IMLFQ should be trained using new type of processes. This method may act like FCFS or RR methods by adjusting the parameters which change the number of queues and the quantum of them. When the fault tolerant mechanism is employed and the first training phase is completed, the network for the other new comer processes works better.

The process service time log files are stored in the system and can be used to compare the average response time. Response time computation using FCFS and RR method shows a run time overhead. If the average response time of FCFS and RR methods, for more than a pre specified percent of instances, is less than the IMLFQ response time; it means that the IMLFQ network needs to be trained using new processes. So we will set the IMLFQ parameters by injection of stored log files. Now IMLFQ will schedule using the new quantum and the new number of queues. This fault tolerant mechanism has minimum software redundancy while the amount of faults in comparing with FCFS and RR scheduling algorithms is considerably reduced. By comparison between IMLFQ and RR we reduce the overhead; it may possible that response time of FCFS is lower than IMLFQ so we check the response time with RR and if the IMLFQ response time is bigger than the FCFS and RR network will be update.

In combinational function we can use any logical function like *AND*, *OR* and etc. Also we can change percentage of FCFS and RR to compare with IMLFQ. Combinational function is kind of filter that it cause reduces the noise in average response time computing, and by do this extra IMLFQ network update will not be occur.

Comparison function compares the IMLFQ average response time with the output of combinational function, that it is combinational average response time.

The fault tolerant architecture is shown in Fig. 2. The pseudo code of fault tolerant mechanism is shown below.

Combinational fault tolerant:

1. Find the average response time of new processes entered in the system using IMLFQ method.

2. Store the log file of processes which processed by IMLFQ method.

3. Compute the average response time for processes that we stored their logs in the system, using FCFS and RR scheduling algorithms.

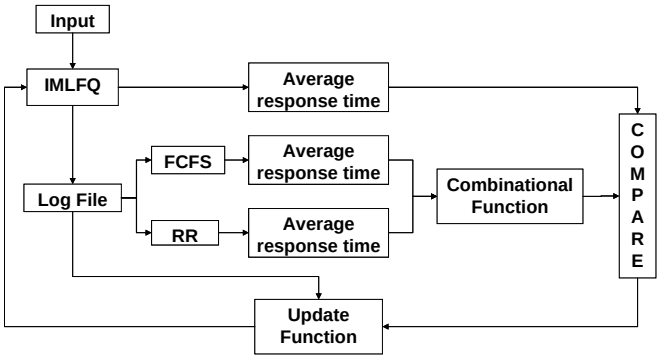

Fig. 2 Fault tolerant architecture for IMFLQ algorithm

4. Compute the combinational average response time by FCFS and RR.

5. Compare the average response time of IMLFQ with combinational function.

6. If the number of repetition is lower than the pre-defined value, go to step 1, else go to step 7.

7. If the numbers of detected faults are more than the pre-specified threshold, update the IMLFQ's parameters.

V. Simulation and Experimental Results

Since the process arrival time is randomly distributed, we used discrete event technique simulation. So the system state has been changed when an event occurred during the simulation time. At first, we sort the processes by their arrival time and then find the first process to handle and provide its service. The process arrival time has a Poisson distribution with an arrival rate of λ=0.8 and service time has an Exponential distribution with mean of μ=0.1. The parameters used in here are the same as those used in Unix-based

operating systems. The simulation consists of 20000 processes with an arrival rate of λ=0.8. When the type of the processes is similar to processes used for IMLFQ training, the IMLFQ average response time is better than the other scheduling algorithms [1].

If the processes service time has a wide range, it affects the network training phase and IMLFQ response time may be more than the other scheduling algorithms. Simulation performed on 20000 processes and the average response time has been calculated for 50 processes at each time. Injected fault are grouped into 10 groups of 50 faults. Obviously the IMLFQ performance has a reduction in compare with combinational function output because of the presence of the faults.

In this paper we consider 30% as the threshold, which means that if combinational function output in 30% of instances is lower than average response time of the IMLFQ, the network will be started to train again. The experimental results are shown in Figs. 3 and 4.

VI. Conclusion and Future Works

The IMLFQ is aimed to present an intelligent algorithm to optimize both the average response time and the waiting time. To do so, the MLFQ has been optimized using RNN and response time has been optimized by learning the neural network. When the response and waiting time optimization is aimed, the IMLFQ shows a good performance, but the learning time of the network is directly related to the amounts of input data, so it is possible to encounter an addition of initial some overflow on the system at the beginning. The combinational fault tolerance method which we presented in this paper leads to the IMLFQ improvement in the average response time. IMLFQ has fault tolerance mechanism so it can detect a fault and improve the average response time simultaneously. IMLFQ can be adopted with any scheduling algorithm using adjustment of its parameters. IMLFQ algorithm with fault tolerant mechanism can be used in real-time system; also this approach is a good scheduling algorithm for interactive systems. We tried to decrease the overhead of the system, however we have a little overhead to be calculated

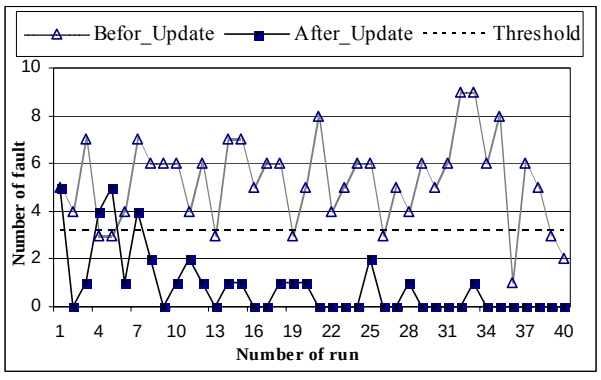

Fig. 3 Comparing between IMLFQ & combinational function before and after the parameters update, that each run include 500 processes

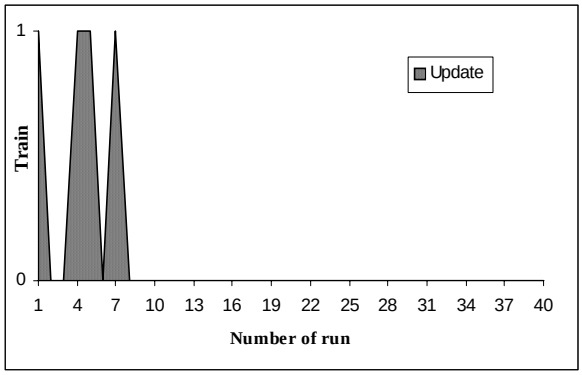

Fig. 4 RNN training after fault detection, that each run include 500 processes

for interactive systems. We tried to decrease the overhead of the system, however we have a little overhead to be calculated and compared with the response time. With more researches it can avoid starvation in IMLFQ. This algorithm can also be used on distributed system, in an effective way that the research in this field is still being continued.

References

[1] M. R. EffatParvar, M. EffatParvar, A. T. Haghoghat, R. Mahini, and M. Zarei, "An Intelligent MLFQ Scheduling Algorithm (IMLFQ)," Real-Time Computing Systems & Applications (RTCOMP), Jun 2006.

[2] K. U. Herath, and Sh. Hashimoto, "Automated trend diagnosis using neural networks," 0-7803-6583- IEEE, 1186-1191, 2000.

[3] C. Molter, U. Salihoglu, and H. Bersini, "Introduction of an hebbian unsupervised learning algorithm to boost the encoding capacity of Hopfield networks," Proceedings of the IJCNN, 2005.

[4] Ma. Sheng, and Ji. Chuanyi, "Fast Training of Recurrent Networks Based on the EM Algorithm. Transactions on Neural Networks," IEEE, Vol. 9, No.1, Jan 1998.

[5] N. K. Kasabov, Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering, A Bradford Book The MIT Press Cambridge, Massachusetts London, England, 1996, Massachusetts Institute of Technology, 1998.

[6] F. A. Gers, and J. Schmidhuber, "LSTM recurrent networks learn simple context free and context sensitive languages," Transactions on Neural Networks, IEEE, 12(6):1333-1340, 2001.

[7] J. Guynes, "Impact of System Response Time on Stat Anxiety," Communications of the ACM, 1988.

[8] A. Memon, A. Porter, C. Yilmaz, A. Nagarajan, D. C. Schmidt, and B. Natarajan, "Skoll: Distributed Continuous Quality Assurance," Proc, Int'l Conf, Software Eng, (ICSE), pp. 459- 468, 2004.

[9] S. Ghosh, R. Melhem, and D. Mosse, "Fault-tolerance through scheduling of aperiodic tasks in hard real-time mul-tiprocessor systems," IEEE Trans, Parallel and Distributed Systems, vol.8, no.3, pp.272-183, Mar 1997.

[10] G. Manimaran, and C. Siva Ram Murthy, "A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis," IEEE Trans, Parallel and Distributed Systems, vol.9, no.11, Nov 1998.

MohammadReza EffatParvar is a student in department of Electrical, computer and IT engineering at Qazvin Islamic Azad University, Qazvin, Iran. He is a member of Young Researchers Club of Islamic Azad University. He gained the second place in Robocup 2005 world cup in Coach Simulation competition. His research interests include NP-Complete algorithms, Data Mining and Artificial Intelligent.