Real-time Scheduling Algorithms: A Survey

Submitted By:

LUO Zijian : A0224725H

LIU Tianshu: A0224716H

Group:33

Supervisor:

Prof. Bharadwaj Veeravalli

Electrical and Computer Engineering, School of Engineering

28 February 2021

Abstract

Real-time system aims to ensure all the tasks are completed by the predetermined time with the logical correct result. In this process, different scheduling algorithms help the system to allocate computation resource and improve its utilization. In this paper four different scheduling algorithms have been presented and compared: Least Slack Time Rate First, Enhanced Maximum Urgency First, Optimized Shortest Job First and Multi-Level Feedback Queue with Intelligent Mitigation. This survey covers the advantages/limitations of each algorithm, as well as the comparison among these algorisms.

1. Introduction

A real-time system is defined as one that responds to and completes the generated tasks within a defined time frame and guarantees the correctness of the logic.

Generally speaking, real-time systems are divided into three types, hard, soft and firm. Hard real-time systems are typically used in the scenarios where the system must respond to immediate events and never miss the event deadline. For example, aerospace field, measurement and control field. Soft real-time is generally used for the tasks where the deadline is not so strict. Occasionally misses the deadline of the task will has little impact on the system. For example: webpage updating, train ticketing selling system, etc. Some scenarios also use firm real-time which allows the task to time out and discard the computation result. There is no doubt that real-time systems are used a lot in people's daily life and are closely related to the development of technology.

Real-time task scheduling theory is the core and key of real-time technology. This is because real-time tasks have time limit requirements. Scheduling real-time tasks among one or more processors requires to determine whether the execution of each task can be completed within the deadline. If the execution of each task can be completed within its deadline, then this scheduling algorithm is said to be feasible. The feasibility is to determine whether the given n tasks can produce a feasible schedule with the application of some scheduling algorithm. The scheduling algorithm is designed to satisfy the task schedulable requirements as much as possible.

In the early days, the real-time systems were relatively simple. The process of scheduling algorithms was quite straightforward for the applications such as microcontrollers, single board computer and some embedded systems. However, as the technology continues to grow and intelligent applications (e.g., artificial intelligence, computer vision) become more and more popular, real-time applications become increasingly complex. The rising volume of tasks and data as well as the increasing complexity of the computation make the traditional scheduling theory difficult to adapt to the needs of today's applications.

More and more researchers start to focus on the updating and developing new scheduling algorithms for real-time system. Based on the different scheduling environments and scheduling requirements, different scheduling algorithms have been developed. In this paper, we present four new scheduling algorithms and compare them scientifically.

Section 2 describes the four new algorithms that we will study and their improvement history. Some well-known algorithms, such as LST, LRT, MUF and SJF will be briefly introduce. Most importantly, these 4 new algorithms will be compared with each other. Depending on their own characters, such as decision method, starvation and waiting time etc, a detailed comparison will be made. Section 3 contains the conclusion as well as the future work.

2.Scheduling Algorithms

2.1 Least Slack Time Rate First (LSTR)

The LSTR is a dynamic priority driven algorithm that can practically assign all the task into processors [1]. It is designed from well-known algorithms : Earliest Deadline First (EDF) , Latest Release Time First (LRT) and Least Slack Time First (LST). LSTR breaks the limitations of the above algorithms and perform a better multi-processor utilization.

EDF gives the higher priority to the task that needs to meet the earlier deadline. The algorithm requires the queue be maintained in the system, which is ordered by the early deadline of each task. When selecting tasks, the scheduler always selects the first task in the ready queue then assign it to a processor and puts it into operation. The earliest deadline priority algorithm can be used in both preemptive and non-preemptive scheduling methods. Under the preemptive uniprocessors' situation, EDF is the optimal scheduling algorithm and it can achieve the 100% utilization bound.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \qquad (1)$$

LRT, which is also called reverse EDF, gives the higher priority to the task that has the later release time. If (1) is being fulfilled, LRT can be treated as the optimal algorithms.

LST which is also called least laxity first, gives higher priority to the smaller slack time. Same as LRT, if (1) is being fulfilled, LST can be treated as the optimal algorithms.

Even though LST, LRT and EDF could achieve the optimal solution, but this only suitable for single processor. If these 3 algorithms are applied to multiple processors, some processors may be left idle, causing the deadline missing.

LSTR is able to overcome the above limitation and is capable to assign all the jobs on uni/multipe processor(s) without idea state.

Firstly, there are 5 prerequisites for LSTR that list below:

CONDITION	PREQUISITE
Preemption	Accept
Migration	Accept
Periodic Task	Only
Release time	0
Timing constraints	Tasks should satisfy (2)

Table 1Prerequisites for LSTR

$$\sum_{i=1}^{n} \frac{e_i}{D_i} \leq n_c \qquad \frac{e_i}{D_i} \leq 1 \qquad (2)$$

Secondly, a special condition called maximum occupation time (MOT) is used to restrict the processing time.

$$MOT = \underset{i \in task\_set}{\arg} \ min(D_i - e_i) \qquad (3)$$

Under the restriction of the MOT, all the job will be released from the processor if the operation time reach the MOT value.

Lastly, to avoid the idle state, the task's execution time will be measured at each scheduling.

$$\text{rate (task }_i) = \frac{e_i^r}{d_i - t} \qquad (4)$$

Under the conditions of all (2) (3) and (4), the LSRT can be the optimal algorithm with multiple or single processor.

2.2 Enhanced Maximum Urgency First (EMUF)

The EMUF is a mixed priority driven algorithm that has the advantages of both fixed and dynamic scheduling [2]. It is improved from well-known algorithms: maximum urgency first scheduling (MUF) and modified maximum urgency first scheduling (MMUF). LSTR breaks the limitations of the above algorithms and perform a better CPU utilization and throughout.

MUF is an algorithm that assign an urgency value to each task. This urgency value is determined by the combination of two fixed priorities and one dynamic priorities. These priorities are assigned to each task via two phases which consider the CPU utilization and the laxity. The importance of the three priorities is as follow:

fixed priority (critical priority) > dynamic priority > fixed priority (user priority)

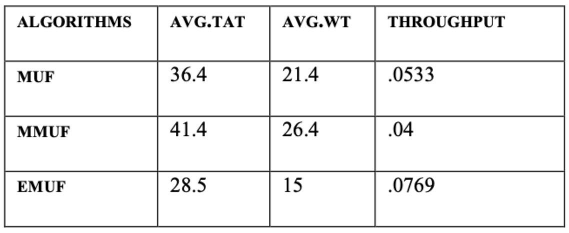
Failure of tasks in the MUF algorithm occurs mainly for tasks that do not enter the critical zone, while scheduling occurs dynamically, and it is unpredictable which tasks can enter the critical zone.

To improve the MUF, the MMUF was proposed [3]. A new parameter which can define the critical level of each task is used in the MMUF. It redefines the critical level as not all the task which with shortest request is the critical one. Here,

fixed priority (user priority)>fixed priority (critical priority) > dynamic priority

However, In MMUF, the CPU utilization will potentially become very low. This is because the system will rank the process with the earliest deadline first, even though the process may miss the deadline.

Similar as MMUF and MUF, EMUF is also a mixed priority algorithm. In the user priority aspect, higher priorities will give the task with more important processed. Differently, here, the intelligent laxity is used to define the dynamic priority. The higher priority will give to the task with minimum intelligent laxity. Here,
fixed priority (critical priority)> dynamic priority >fixed priority (user priority)



ALGORITHMS	AVG.TAT	AVG.WT	THROUGHPUT
MUF	36.4	21.4	.0533
MMUF	41.4	26.4	.04
EMUF	28.5	15	.0769

Table 2 Comparison Between MUF, MMUF and EMUF

These are the different results produced by different algorithms for the same task. Compared to MUF and MMUF, EMUF has reduced both turnaround time (AVG.TAT) and waiting time (AVG.WT).And the throughout has been increased. The detailed information will be presented in the CA2.

2.3 Optimized Shortest Job First (OSJF)

This optimized algorithm is based on the smallest jobs scheduled algorithm (SJF). In this algorithm, the order of the process in the queue list is dependent on the burst times of the processes. The process which has the shortest burst time is ordered the first position in this queue list. Therefore, when the CPU is ready, the first process in the list is selected and then transmits to the CPU for execution. After finishing the execution of this process, the queue list removes it. And then, the next top process is selected from this waiting list.
In order to minimize the context switching, the optimized shortest job first scheduling algorithm is invented. For the innovation of

this proposed algorithm, the most impressive advantage is conditional preemptive. Especially for a new job, it is very essential to decide whether to preempt the running process or not. Obviously, the newly arrived process will not preempt when its burst time is more than the remaining execution time of the running process. But for the opposite condition, the paper told us there are two possibilities. The first one is that the running process will not preempt, when half of the remaining execution time of the running process is less than the burst time of the newly arrived process. And the second one is that the processor will preempt the running process and then execute the newly arrived process, when half of the remaining execution time is greater than the burst time of the new process.

From the experiment in this paper, we can get the conclusion that there is a great improvement at sufficient reduction in context switching. The decrease of average waiting time is so small, but compared to other traditional scheduling algorithms, this optimized shortest job first scheduling algorithm is still better in average waiting time.

And then, when we compare to STCF (Shortest Time-to-Completion First) algorithm, it is truly preemptive when a newly arriving process into the queue list. It is possible to let this operation system more and more congested. Therefore, if the processor determines the preemptive activity by conditions, this problem is easily solved.

Within the analysis of scheduling algorithms, we can find that the most advantage of the traditional SJF algorithm is less average waiting time for all available processes. For the proposed algorithm, inherits this advantage and makes a progress in reducing context switching. However, there is still a serious problem. The processes with large execution time must wait for a long time to get their turn when short processes arrive continuously. We call this condition is starvation.

2.4 Multilevel feedback queue-Intelligent mitigation

This algorithm is innovated from traditional Multi-level Feedback Queue. As for the basic rules of traditional MLFQ, there are a amount of queue lists that they are set a various priority level. Only one task is allowed to run in a single queue at any given time. The order of process in waiting list is dependent by the priorities. Therefore, processor will choose a job with higher priority to run after the running process is finished. And there are five basic rules about MLFQ [4].

Rule1: If Priority(A) is higher than Priority(B), B have to wait after A is executed fully.

Rule2: If Priority(A) equals Priority(B), the CPU use round-robin method [5] to run A and B in any queue level.

Rule3: The operation system will put the newly arrival process at the highest priority.

Rule4: If a job drains up appropriate time allotment, the processor will reduce its priority step by step.

Rule5: To reduce the starvation, the processor moves jobs with lower priority to the higher priority queue, in any period of time S.

The most advantage of MLFQ is that it monitors the execution of a job and gives process updated priority accordingly, instead of presetting fixed priority of many processes. CPU comprehensive process will drop down through the queue waiting list until it is turn of the final queue. As for the basic part of traditional one, it has two advantages. The first one is the optimization in turnaround time. And the another one is minimization of response time, which means it allows operation system to be more responsive to interactive users.

As for the MLFQ-IM, it is investigated from MLFQ in real time environment. The first change is that starvation mitigation, which means this optimized method arrange some CPU time to run a number of processes with the lowest priority. As we all know, in heavy processing load, some jobs with lowest priority can be temporarily starved for a long time, because the processor always run the higher priority process firstly at any given time. In order to reduce starvation, it is valuable to redirect the two lowest priority queues, Qn-1 and Qn. The second improvement is the updated priority in Q1 by monitoring. If there is sufficient time to be used in Q1, the processer will release a part of time to lowest priority queue. It is so important to mitigate the starvation in lower priority queue and enhance the throughput of total operation system.

Compared to same type MLFQ algorithm, in lowest priority queue, we can easily find that MLFQ-IM performs significantly better than MLFQ and MLFQ-NS [6], especially on the condition of over 100% of CPU capacity. For the lower priority, when it is below 140% capacity, the traditional MLFQ algorithm perform better than MLFQ-NS and MLFQ-IM, but when it exceeds 140 % capacity, the starvation of MLFQ-IM in lower queue is advanced than others. When it comes to the CPU redirection in Q1, through the experiments, it is obvious that mean waiting time in Q1 are not affected by basic mitigation strategies. Especially for the effect of total throughput, we can conclude that it is effective and efficient to obtain a higher system throughput even under an appropriate heavy load.

To sum up, the extension of MLFQ, which is MLFQ-IM, and concluded that there is a range of system load percentiles in which usage of MLFQ-IM is appropriate. Burst time in Q4 and Q5 increased greatly, with no apparent effects in Q1. However, it is bad result when the system load is over 140%. In other words, the MLFQ-IM is appropriate to use in less 140% loading condition. When it comes to extremely heavy load, it still performs worse in some aspects.

2.5 Comparative Analysis of scheduling algorithms

Table 3 below shows the comparison among the 4 different algorithms.

	EMUF	LSTR	MLFQ-IM	Optimized SJF
Priority	Fixed and Dynamic	Dynamic	Dynamic	Mainly Dynamic
Context Switching	Small	Small	Small	Small
Decision Method	Urgency based	Deadline based	Preemptive	Conditional Preemptive
Response time	Low	Low	Low for I/O process	Low for short process
Waiting time	Low	Low	Low for largest priority	Low

Throughput	High	High	Very high in appropriate system load	High
Starvation	Less	Less	Less	High
Predictability	Predictable	Predictable	Not predictable	Predictable
Effectiveness	Efficient to obtain a higher system throughput	Optimal possibility for both uni- and multi-processor environments	Efficient to obtain a higher system throughput	Reducing context switching
Limitations	Not stated	Time loss is slightly longer than EDF	Starvation still exists in heavy load	Starvation condition exists
Multi-Processor	Suitable	Suitable	Not suitable	Not suitable

Table 3 Comparison of four scheduling algorithms

Priority: The type of priority.

Context Switching: The number of times that a task is pre-empted by a higher priority task.

Decision Method: Design principles/basis of the algorithm.

Predictability: Predictability means that the algorithms are able to meet mission-critical timing requirements with 100% assurance over the life of the system, to evaluate the overall performance of the system over different time frames. At the same time, the algorithm is capable to evaluate the performance of individual tasks at different times and as a function of the current system state. If the timing requirements are met by these evaluations, with respect to its timing requirements, the system is predictable.

Effectiveness: Factor of good result.

Multi-Processor: Whether the algorithms can be implemented under the multi-processor environment.

From the table we can tell that all these scheduling algorithms are dynamic priority based. In other words, during the period of scheduling, the processor updates the priority order of the process at any time to achieve the optimal strategy. For the EMUF scheduling algorithm, it is very useful to provide more and more task-oriented results. But when it turns to LSTR, this algorithm sufficiently reduces the context switching and it is also appliable in multi-processor conditions. As for MLFQ-IM, the most valuable point is that it prevents serious starvation in some aspects and gets a higher system throughput. Lastly, for the optimized SJF algorithm, easily justifies whether to preempt in two possible conditions. At the same time, it reduces the time in context switching.

3.Conclusion

Real-time system become more and more popular. In the real-time system, the performance depends not only the time instance at which the reactions are made, but also on the logical computations.

In the LSTR paper, the authors bring a new algorithm that could run in both uni-processor and multi-processor to us. The optimal possibility have been showed clearly through the experiments. However, the time loss is longer than EDF, which could be improved in the future work.

For the EMUF paper, the authors show us the limitations of the MUF in detail as well as the continuous enhancements and developments for this basic algorithm (from MUF to MMUF, from MMUF to EMUF). This step-by-step idea can inspire future improvement on the similar algorithm.

In the OSJF paper, the author just updates the preemption condition, which truly increases the total performance through comparing results. However, it is still not easy to detect the position of every preemption. That is to say, the preemption cost is not easy to justify. For the author, this is a place that lacks consideration.

From the simulation experiments, the MLFQ-IM algorithm clearly shows its flexibility in heavy load endurance and effectiveness in mitigation starvation in different queue levels. We can easily find it performs worse in the prediction of starvation. Especially when the system load is large, it is very hard to determine which queue level happens starvation. Therefore, there is a lot of future work to finish.

Although the new algorithms still have limitations, all of them overcome the shortcomings of the previous algorithms and achieve improvements in performance. We believe more and more excellent algorithm will be discovered in the future.

4.Reference

[1] M. Hwang, P. Kim and D. Choi, "Least Slack Time Rate first: New Scheduling Algorithm for Multi-Processor Environment," in *2010 International Conference on Complex, Intelligent and Software Intensive Systems*, 2010.

[2] B. H.S., N. Raffat and M. Mallik, "Enhanced Maximum Urgency First Algorithm with Intelligent Laxity for Real Time Systems," *International Journal of Computer Applications ,* vol. 44, 2012.

[3] V. Salmani, S. T. Zargar and M. Naghibzadeh, "A Modified Maximum Urgency First Scheduling Algorithm for Real-Time Tasks," *World Academy of Science, Engineering and Technology,* 2005.

[4] S. Kadry and A. Bagdasaryan, "New Design and Implementation of MLFQ Scheduling Algorithm forOperating Systems Using Dynamic Quantum," *STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING,* vol. 3, pp. 189-196, 2015.

[5] A. Singh, P. Goyal and S. Batra, "An Optimized Round Robin Scheduling Algorithm for CPU Scheduling," *International Journal on Computer Science and Engineering,* vol. 02, pp. 2383-2385, 2010.

[6] E. J. Brown, "On Intelligent Mitigation of Process Starvation In Multilevel Feedback Queue Scheduling," Kennesaw State University, 2016.