Advances in Computer Science and Information Technology (ACSIT)
Print ISSN: 2393-9907; Online ISSN: 2393-9915; Volume 2, Number 11; April-June, 2015 pp. 12-15

http://www.krishisanskriti.org/acsit.html

UPD- Static Priority Real-Time Algorithm for Multiprocessors based on RM

Faiyaz Ahmad[1] and M. Nauman Siddique[2]

[1,2]Department Of Computer Engineering, Faculty of Engineering & Technology, Jamia Millia Islamia New Delhi -25, INDIA
E-mail: [1]ahmad.faiyaz@gmail.com, [2]naumansiddiqui4@gmail.com

Abstract—*In past few years the real time systems have seen a shift from uniprocessors to multiprocessors. Uniprocessors have optimal algorithms RM, EDF and LLF but these algorithms prove to be suboptimal for multiprocessor real time systems. Multiprocessor real-time systems require optimal algorithms to meet their logical correctness in constrained time.*
Several algorithms have been proposed based on RM, EDF and LLF in past few years. In this study, we propose UPD based on RMZLPD and RM. Through simulation, UPD has shown high success ratio and schedulability ratio gain over its predecessors. UPD being a static priority algorithm aims at meeting the deadlines of high priority tasks even under overloaded conditions. UPD has better success ratio and schedulability over algorithms taken for consideration.

Keywords*: real- time scheduling algorithms, RMZLPD, UPD, CPU Utilization Rate, success ratio, schedulability*

1. INTRODUCTION

The basic characteristic of a real time system is logical correctness in constrained time. A time – critical task has a predefined deadline. The scheduling algorithms specify the order in which the tasks are to be executed. Time-critical tasks are periodic and need to be completed before its deadline is reached. They are independent and do not depend on any task for initiation and completion. [2]

Based on scheduling table and schedulability analysis, real time systems are –Dynamic and Static Real Time Systems. Dynamic real time systems assign priorities to tasks on any criterion while static real time systems have user defined priority. Based on the preemption of tasks, real time systems are–Preemptive and Non-Preemptive Real Time Systems. Preemptive real time systems allow the lower priority tasks to be preempted for any higher priority tasks. Non-preemptive real time systems do not allow preemption of tasks until they are finished. [3]

Multiprocessors real time systems are divided in two paradigms – global scheduling and partioned scheduling. Global scheduling allow tasks to run on any processor without any processor affinity while in partioned scheduling tasks run on the processors assigned to them beforehand. Partioned scheduling is relatively easier to implement but global scheduling has better resource management and is more robust.[1] The task distribution for partioned systems are usually done manually or by non-optimal heuristic techniques.[5] When tasks are partioned and scheduled they are more prone to unbalanced load distribution and have higher preemptions than a globally scheduled system.[4]

It has been proved that the tasks systems that are schedulable under partioned approach are not mandatory to be schedulable under global approach. Also, the task systems that are schedulable under global approach cannot be partioned into subsets. [6]

We propose global static scheduling algorithm using the concepts of RMZLPD [1] and RM. RM schedules tasks with least response time but we consider CPU Utilization Rate over response time and draw the concept of pseudo-deadline from RMZLPD. In our study we assume that processes with higher utilization rate are of high priority and need to be completed before its deadline over other low priority tasks. The tasks with high priority have hard pseudo-deadline requirements so as not to miss their deadline by any chance but the tasks with low priority have soft pseudo-deadline and can miss their deadline at the cost of any high priority task.

2. SYSTEM MODEL

We consider periodic tasks τ_1 , τ_2 ……τ_n of the form τ_i= (C_i ,D_i) where C_i denotes computational cost of the deadline of the task and D_i denotes the deadline of the task. The CPU Utilization Rate of a task is calculated by U_i= C_i \ D_i where i= 1 to n. The system utilization rate is calculated by $U=\Sigma_{\tau k}{}^{\tau n}$ U_k/m where m denotes the number of processors. For a task set to be schedulable, U must be less than or equal to 1.

3. RELATED WORK

RM (Rate Monotonic)

RM is global preemptive fixed priority scheduling [1]. RM assigns high priority to tasks with lower computation cost than tasks with higher computation cost. The task with shortest computation cost is scheduled first.

Theorem 1: If $U \leq n (2^{1/n} -1)$, n independent periodic tasks can be scheduled by RM.[2]

In theorem 1 when $n \rightarrow \infty$, $U \approx 0.693$.This implies the maximum CPU Utilization Ratio of RM is 69.3. When the load is greater than U, RM cannot schedule those tasks. RM is an optimal algorithm for uniprocessor systems but not for multiprocessor systems. The advancements made in the algorithm for increasing U are sub – task method and dual – priority method.

Sub-Task method: Tasks are divided into consecutive sub – tasks. Every sub – task has its own priority and their priorities are non – descending. In this method the task that initially had low priority will gain a higher priority after certain time periods due to the execution of the other high priority tasks. [7]

Dual – Priority method : Period of the tasks are divided into two stages which has a different priority .Similar to sub- task method it is possible that the lower priority task in its second stage could preempt the task which has the higher one.[8]

RMZL

RMZL is based on RM with an additional feature of zero laxity. In RMZL, jobs are scheduled in accordance with fixed priority of their associated tasks, until a situation arises where the execution time of a task is equal to the time of its deadline. A job with zero-laxity will miss its deadline unless it executes continuously till completion.[1]

RMZL has work conserving and domination property. Work conserving property means the processor never remains idle as long as jobs are in the ready queue. Thus, maximizing the average response time of the system. Domination property states that RMZL has all the properties of RM in normal circumstances but overrides them once zero laxity occurs. [10]

RMZL gives higher priority to tasks with zero-laxity. The task scheduling in RM and RMZL are identical until a zero-laxity job arrives. This shows that RMZL is superior to RM as it can schedule tasks feasible by RM and also schedule tasks with zero-laxity.

EDZL

This algorithm uses EDF as long as no zero – laxity tasks occur. When a task with zero laxity occurs, it preempts the task with higher deadline amongst the currently executing tasks. When a tie occurs choose the task with lowest computation cost. [9]

LP-RMZL

LP-RMZL is based on RMZL. In LP-RMZL the tasks with higher priority cannot preempt a task of lower priority except for by the zero-laxity tasks. This implies a task once assigned processor cannot be preempted until a zero-laxity task arrives. LP-RMZL has lower task switching and better success ratio than its predecessor RMZL. [1]

RMZLPD

RMZLPD adds an additional feature of pseudo-deadline to RMZL. In RMZLD, tasks are scheduled according to RMZL, until a situation arises where the remaining pseudo execution time of a job is equal to its pseudo deadline. Such a job has pseudo zero laxity and will miss its deadline unless it executes continuously to its pseudo deadline. [1]

RMZLPD gives semi highest priority to jobs with pseudo zero laxity until its deadline. Pseudo deadline is calculated by setting the deadline to its half. RMZLPD has higher schedulability ratio over its predecessors.

Pseudo deadline = Deadline \ 2

Half Execution Rate = Computation \ 2

4. PROPOSED ALGORITHM

UPD

We have proposed an algorithm UPD CPU utilization based pseudo deadline algorithm. Under UPD jobs are scheduled according to their CPU utilization rate, until a situation arises where the remaining execution time of the task is equal to the time of its deadline. Such jobs having zero laxity will miss their deadline if they are not granted CPU execution time. UPD calculates pseudo deadline equal to half of its deadline. A task with high CPU utilization rate is given higher priority over lower priority tasks. Now to avoid lower priority tasks missing their deadline, they are given high priority as soon as they reach their pseudo deadline over all other high priority tasks. Tasks having equal CPU utilization rate are resolved by assigning high priority to tasks having lower deadline values.

Algorithm for calculating Pseudo Deadline

Input: Tasks in the form (C_i, D_i)
Output : Pseudo Deadline and Half Execution Rate for tasks

```
If(Deadline is odd for a task)
{
Pseudo-deadline = (Deadline / 2 ) +1
Half execution rate at Pseudo-deadline = (C /2)+1
}
Else
{
Pseudo-deadline = (Deadline / 2 )
If( Computation cost is odd)
Half execution rate at Pseudo-deadline = (C /2 )+1
Else
Half execution rate at Pseudo-deadline = (C /2 )
}
```

Algorithm UPD

```
While (True)
{
If(Clock = 0)
Schedule m tasks with highest CPU Utilization Rate
Else
{
if (Pseudo-Deadline for any process = Clock)
Schedule it replacing it with current active processes
If(Half Execution Rate for currently active processes = 0)
Schedule the next highest priority task
Else
Exhaust the clock cycle by the currently active processes
}
}
```

5. EXPERIMENTAL SETUP

We describe the overall structure of our experiments. We have chosen RMZL, LPRMZL and RMZLPD as the algorithms under consideration. We simulate each algorithm for obtaining success ratio and schedulability rate for the tasks sets provided to them and comparing these results with those of UPD. For each consideration we take 1000 randomly generated task sets having CPU utilization rate lying in range of (0, 1) and to have multiprocessing effect we execute it simulating 4 processors. Each task has its computation cost and pseudo deadline values. We compute CPU utilization rate of each process and schedule them on the basis of their priorities. For simulation results we have considered the interval of CPU Utilization Rate [0.4 , 1) with a step size of 0.04 because from 0 to 0.4 the algorithms considered did not show deviation from normal, proving fruitless for our experimental result. But for intervals 0.4 to 1 we can see the deviations in the graph of algorithms under consideration in our simulation results.

6. RESULTS

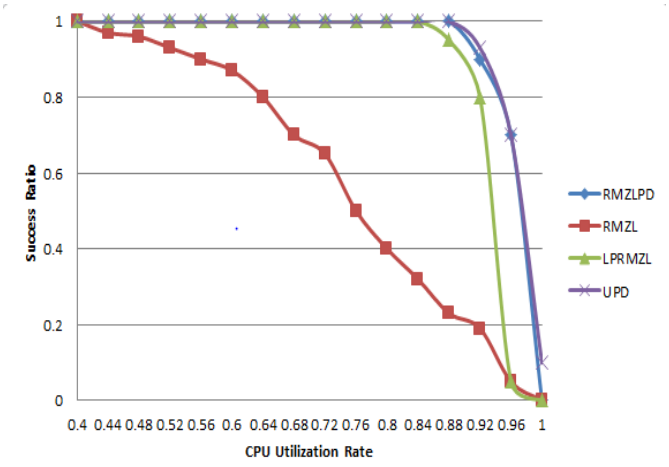

Fig. 1: Result of Success Ratio

We have simulated the randomly generate task set on RMZL, LPRMZL, RMZLPD and UPD. The tasks sets in RMZL were scheduled similar to RM except when any zero laxity error was encountered. The tasks in LPRMZL were scheduled similar to RMZL and the tasks couldn't be preempted until a zero laxity task. RMZLPD scheduled the tasks similar to RMZL to meet their pseudo deadlines.

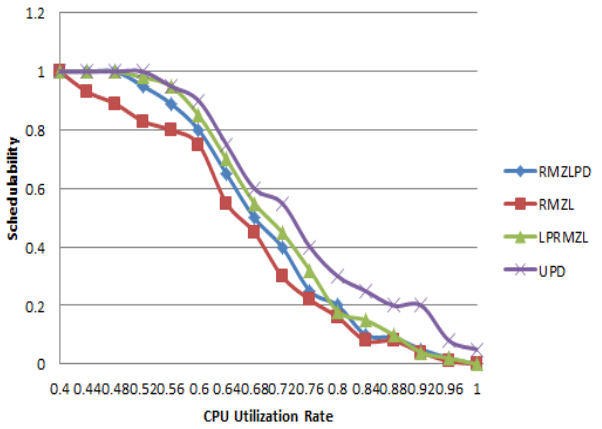

Fig. 2: Result of Schedulability

Tasks under UPD were scheduled on the basis of their CPU Utilization Rate. The initial step was to calculate the pseudo deadline of each task. The tasks with deadline in even clock cycles had their pseudo deadline and half execution rate equal to half of their deadline and computation cost values respectively. But the tasks with odd clock cycle value of their deadline had pseudo deadline value equal to the ceil value of deadline. Now if computation cost had even value the half execution rate was set to half of computation cost else to the ceil value of the computation time. Now the tasks with highest values for CPU Utilization Rate were scheduled but if two tasks had equal utilization rate then the task with lower deadline value was given the higher priority. The tasks under UPD have more process migration than RMZLPD but the consideration to meet the deadline of any task is much greater than the time taken in context switching so we have not considered this aspect of the task.

Figure 1 and figure 2 show the results of our simulation in the range (0.4 , 1]. Figure 1 shows the results of our simulation for success ratio. It shows a better success rate over its predecessors RMZL, LPRMZL and RMZLPD. As in figure 1 we can see the success ratio UPD is near the threshold mark proving its high success ratio for the scheduled tasks. The success rate of UPD results show a 100 % in the range of CPU utilization rate value (0 ,0.9) proving it to be an algorithm with high success ratio. The results show that the algorithms RMZLPD and UPD have a much better success rate than RMZL and UPD.

Success rate is the ratio of tasks can have been scheduled by the algorithm to the total number of tasks submitted to the

processors. An algorithm with success rate can be considered as more optimal algorithm. Schedulability is the capacity of an algorithm to finish the tasks within their constrained deadline clock cycles. The two factors success ratio and schedulability of any algorithm are the best parameters to test the fitness of any algorithm on any set of task set.

Similarly figure 2 shows the results of simulation with schedulability are highly encouraging over its predecessors. The graph for RMZLPD and RMZL shows a dip for CPU utilization rate values for overloaded conditions but UPD performs better than it. The graph for UPD shows better results which is a better sign for future developments in the algorithm. The UPD algorithm under any circumstance serves its purpose of scheduling high priority tasks even if it might miss the deadline for low priority tasks.

7. CONCLUSION AND FUTURE DIRECTION

The simulation results of UPD show its high success rate and an enhancement over its predecessors in schedulability rate. UPD can calculate pseudo deadline and half execution rate for all the tasks submitted to it unlike RMZLPD which can calculate only for even values of the tasks. UPD can achieve high amount of parallelism and can be an optimal algorithm for priority driven soft real time systems where the deadline of low priority tasks can be missed at the expense of scheduling high priority tasks. UPD has a dynamic approach also, as it can be seen it can assign high priority to tasks that have zero laxity. The algorithm is however complex and requires lots of computations but instead has a way better result than its predecessors.

The results of simulation show a big gap for schedulability owing to the task complexity and assumptions in our approach. We plan to further enhance this algorithm by taking other real time factors into consideration and analyze its schedulability by RTA [4]. We also intend to remove the process migration problem in our next algorithm to make less cumbersome for limited resource systems.

REFERENCES

[1] Yanai K. Yoo M. and Yokoyama T. A Proposal of Real-Time Scheduling Algorithm based on RMZL and Schedulability Analysis. *17th Asia Pacific Symposium on Intelligent and Evolutionary Systems, 2013 ; 9-14*

[2] Dhall, S.K. and Liu, C.L. On a real-time scheduling problem. *Operations Research,* 1978 ; 127-140

[3] Jie, L. Ruifeng, G. and Zhixiang, S. The Research of Scheduling Algorithms in Real-time System. *International Conference on Computer and Communication Technologies in Agriculture Engineering,* 2010 ;333-336

[4] Bertogna, M. and Cirinei, M. Response-Time Analysis for globally scheduled Symmetric Multiprocessor Platforms. *28th IEEE International Real-Time Symposium*,2007 ; 149-158

[5] Ramamurthy, S. and Moir, M. Static-Priority Scheduling on Multiprocessors . IEEE 21st *Real Time Symposium* ,2000; 69-78

[6] Lehoczky, J. Sha, L. and Ding, Y. The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behaviour. *In the proceedings of the10th IEEE Real Time Systems Symposium,* 1989 ;166-171

[7] Harbour, M.G. Klein M.H. and Lehoczky. Fixed Priority Scheduling of Periodic Tasks With varying Execution Priority . *IEEE 12th Real-Time systems Symposium,*1991 ;116-128

[8] Bums, A. and Wellings, J. Dual Priority Assignment :A Practical Method for Increasing Processor Utilization . IEEE 14th Real-Time Systems Symposium , 1993; 48-53

[9] Lee, S. K. On-line Multiprocessor Scheduling Algorithms for Real-Time Tasks. *TENCON '94. IEEE Region 10's Ninth Annual International Conference. Theme: Frontiers of Computer Technology, 1994 ; 607-611*

[10] Kato, S. Takeda, A. and Yamasaki, N. Global Rate-Monotonic Scheduling With Priority Promotion. *IPSJ Transactions on Advanced Computing System*, 2008 ; 64-74