Performance And Deadline Based Real Time Scheduling Algorithm

Laxmi G Jadhav¹, Dr. Radhakrishna Naik²

¹PG Student, CSE, MIT, Aurangabad, Maharashtra, India
²Prof. and Vice Principal, MIT, Aurangabad, Maharashtra, India

Abstract— In real-time systems tasks should be completed and systems goal should be achieved that too within prescribed deadline. Along with deadline contribution of each task could be a major factor during execution of tasks. And performance of each task depend upon the total number of instructions have successfully executed of that task. Because it directly affects the overall performance of process. Now a day’s multicore processor platform and advanced techniques are booming in the market. This paper presents the scheduling algorithm for real-time task set which takes deadline as well as contribution of each task into account while scheduling the task set and paper targets the multi-core processing platform. Simulating the proposed scheduling algorithm through the analysis of four case studies and gives optimal improvement solution for requirement of execution time, turnaround time and bandwidth are less for executing an application on core. Proposed scheduling algorithm is analyzed by simulating real-time task set on identical quad core processor and results are compared with IRMS scheduling algorithm.

Keywords— Real Time Scheduling, Performance Contribution Factor (PCF), Improved Rate Monotonic Scheduling (IRMS), Rate Monotonic (RM) and Earliest Deadline First (EDF), Multicore, Rbound.

I. INTRODUCTION

The distinguished characteristic of a real-time systems compared to general purpose systems is of timing constraints. Real-time systems have to meet many demands in prescribed time known as deadline. Along with deadline task is characterised by its ready time, execution time, performance and resource requirement. Each task contributes to the output of system in some manner so contribution of each task i.e. performance contribution factor should be taken into account. And performance of any task is depend upon the successfully execution of that task. Task successfully executes when number of instructions in that have successfully executed.

Scheduling policies are also greatly affected by the processing platforms. Since the first results published in 1973 scheduling algorithm by Liu and Layland [1] namely Rate Monotonic, Earliest Deadline First are optimal algorithms on uniprocessor processing platform. To acquire highest performance multicore platforms booming in the market. Multicore platform provides the solution to increase the processing speed and it can run many processes parallely in other hand If only operating frequency of single processor increase then it would cause the serious heating problems and the problem of power consumption. Heterogeneous cores also help to increase the performance of multicore processors [2]. To manage the complexity of heterogeneous cores, os should be design properly.

This dissertation suggests preemptive scheduling algorithm and presents the scheduling algorithm for real-time task set on the basis of their deadline as well as performance Contribution of each task taken into account while scheduling the task set. Higher the performance of the task will get higher priority for execution. In MSS it may happen that some task may not perform to their fullest, but contributes up to certain extent or fails completely. It checks not only deadline but
contribution of each task in the final result and accordingly it assigns the task. Task can execute of its total instructions is taken as contribution factor. The rate of performance of task is calculated by calculating the failure probability of each task which is total lost instructions. It is calculated from total number of instructions for execution and how many instructions are successfully executed. This achievement level of each state is assumed.

In the task set task priorities are allocated on the basis of two factors such as Performance Contribution factor (PCF) and Deadline of each task in task set. Result is compared with the IRMS algorithm [5] on uniprocessor. Which used the Rbound technique to schedule the task on uniprocessor as well as on multiprocessor to achieve the higher utilization.

**Organization of the Paper:**

The rest of the paper is organized as follows. Section II represents the related work. Section III represents the system model with terminology and recalls the some background concepts with related work and proposed work for scheduling the application according to the precedence relation along with considering the PCF and Deadline. Section IV illustrates the performance evaluation of the proposed algorithm. Section V represents the performance analysis of the proposed work. Section VI states our conclusion and possible extensions for a future work.

II. RELATED WORK

In real-time systems deadline is the prime constraint along with performance of the task. So there should have an appropriate execution of task. Many researchers and academicians has been done lot of work right from first result published in 1973, by Liu and Layland [1] in the field of real-time scheduling. In the first paper of real-time scheduling Liu and Layland presented optimum fixed priority algorithm i.e. Rate monotonic (RM) and Earliest Deadline First (EDF). In RM algorithm priorities are assigned to task in the task set on the basis of their period whereas in EDF priorities are assigned to tasks in task set on the basis of their deadline.

Yifan Wu, Zhigang Gao [4] addressed Deadline and Activation Time Assignment for Partitioned Real-Time Application on Multiprocessor Reservation they focused on partitioned algorithm to schedule the real time task. Sylvain Lauzac, Rami Melhem and Daniel Mosse [5]. Presented Rbound technique to schedule real time task on uniprocessor as well as multiprocessor to achieve higher utilization. Uniprocessors gave place to multiprocessors since the first microprocessor which was 4 bit by Intel in 1971[6, 7], Paradigms in processing platforms have been shifted over time. If Moore's law is to believed performance will grow every 18 months since transistors on chips will double every 18 months [8]. The performance of using multi-core processors, however, depends on the nature of the applications. Radhakrishna Naik, R.R. Manthalkar, [10] they calculated instantaneous utilization of each task and according to higher utilization of task they scheduled that task . In global scheduling algorithm single ready queue is maintained and task can be migrated or if needed can be scheduled on single core [11]. Even if task is pre-empted on one core can resume its execution on other core.

2.1 Real-time Scheduling on Multi-core Processor Platform

Recent trends revolve around multi-core processors. The performance of using multi-core processors, however, depends on the nature of the applications. To take advantage of the parallelism offered by a multi-core architecture, appropriate algorithms have to be used. Moreover, since processing units i.e. core are located on the same chip and generally have shared memory, communication between cores is very fast. Multicore may be homogeneous or heterogeneous. Algorithm is categorised in two categories one is global scheduling where there is only one global queue [11]. and other is partitioned scheduling algorithm where there are different queue for each processor.
There are various ways to assign task to Multicore to increase the performance of system such as to reduce waiting time, increase the utilization. Various parameters can be focused to design scheduling policy such as cache miss, context switch \[12\] to accomplishing goals.

2.2 Performance contribution of task

Contribution of each task decides the Performance of the system. In some real-time systems, each task contributes to the final logical results. In MSS it may happen that some task may not perform to their fullest, but contributes up to certain extent or fails completely. We have to consider the failure probability of each task it is depend upon the number of instructions successfully executed. PCF of each task can be calculated as

\[ E_G = \sum_{k=0}^{k} p_k G_k \]  

Where,

\( E_G \) = Expected MSS performance.

\( G_k \) = Performance rate at state k of MSS.

\( p_k \) = Failure Probability of system being in state k

The probability that task reaches particular state is calculated by

\[ p_k = Na_{lost} = Na_{in} - Na_{out} \]  

Where,

\( Na_{in} \) = Total number of instructions for execution.

\( Na_{out} \) = Total of number of instructions successfully executed.

\( Na_{lost} \) = Total of number of instructions fails during execution

We can find out performance rate of each task by using

\[ G_k = 1 - p_k \]  

2.3 Challenges for assigning task to Multicore system

If required resources are not available it causes Increases in waiting time which degrades the performance of task Transient Faults have a number of sources that include changes in the operational environment, simultaneous arrival of asynchronous events, system exceptions, and hardware oriented faults in the CPU peripheral devices related to inaccuracy in electromechanical devices, and power fluctuations Due to increase in input traffic congestion will occur will cause performance degradation. Heterogeneous core can cause the problem of speed mismatch

III. PROPOSED WORK

3.1 System Model : In real time, task are scheduled according to precedence constraint due to that higher priority task will get less significance and lower priority tasks will get execute. To avoid such problems system should be model properly. Modelling helps to understand complex problem and their solution \[13\]. Modeling real-time systems can be focused on two parameter correctness and efficiency when processing platform like multi-core, provides parallelism \[15\]. Modeling inter-task communication mechanisms also have been studied in literature \[16\] Systems engineered by
modeling the system in design phase refer to model-driven-engineering. MDE was introduced by Kent [17]. In proposed algorithm tasks are classified according to performance as well as deadline.

3.2 Architecture: In this section describes the how the proposed algorithm works. This dissertation proposes scheduling algorithm for periodic tasks, which can be scheduled on homogeneous multi-core processors OS in non-preemptive manner. Proposed algorithm is hybrid algorithm which classifies the tasks of task set in four classes and determines strategy for allocating task to eligible processing unit i.e. core on multi-core processor. The general view of scheduling policy is as follows

![Block Diagram](image)

As shown in Fig. 3.3 system is modeled first calculate the performance of each task. After calculating performance sort these task in descending order means greater the performance will get higher priority. Then sort the task again in ascending order of deadline, now after sorting the tasks apply proposed algorithm and classify tasks in four classes. After classification assign tasks of class 1 to the all cores parallely. Then assign tasks of class 2 parallely to cores in ascending order when class 1 scheduled same procedure use for remaining classes. Therefore PCF of each task can be calculated as

\[
E_G = \sum_{k=0}^{k} p_k G_k
\]

\(E_G\) = Expected MSS performance.

\(G_k\) = Performance rate at state k of MSS.

\(p_k\) = Failure Probability of system being in state k
The probability that task reaches particular state is calculated by

\[ p_k = Na_{\text{lost}} = Na_{\text{in}} - Na_{\text{out}} \]  \hspace{1cm} (5)

\( Na_{\text{in}} \) = Total number of instructions for execution.
\( Na_{\text{out}} \) = Total of number of instructions successfully executed.
\( Na_{\text{lost}} \) = Total of number of instructions fails during execution

We can find out performance rate of each task by using

\[ G_k = 1 - p_k \]  \hspace{1cm} (6)

Classification of task into two classes on the basis of the two parameters PCF and the Deadline.

**Table 3.1: Classification based on PCF and Deadline**

<table>
<thead>
<tr>
<th>Priority Level</th>
<th>PCF ((E_c))</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 1</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Class 2</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Class 3</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Class 4</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>

3.3 Scheduling Strategy

Assignment of task to the core is done at run time depending upon precedence of task and availability of processing unit \( i.e. \) core. Decision regarding allocation of task to eligible core is taken on the basis of scheduling strategy by the scheduler. Once task is allocated to particular core, task is not migrated to any other core. All the tasks are executed in preemptive manner which means higher priority task are scheduled first, they may not need to wait in queue.

3.3.1 Assumptions

- Only one instance of each task is executed per period in non-preemptive manner.
- All cores are homogeneous \( i.e. \) identical.
- Failure probability should not be greater than 40%.
- All the tasks are single threaded.
- All tasks are independent.

3.3.2 Proposed Algorithm

1. Find performance contribution factor of each task by using following formula
2. Sort the task according to descending order of \( E_c \).
3. Divide the task into two groups. Add flag 1 to first group and 0 for the other
4. Sort the task according to ascending order of deadline \( D_k \).
5. Divide the task into two groups. Add flag 1 to first group and 0 for other as
6. Classify tasks in 4 classes
7. if (Tk .Eflag= Tk .Dflag=1) Then Tk .class = 1
8. if (Tk .Eflag=1&& Tk .Dflag=0) Then Tk .class = 2
9. if(Tk .Eflag=0 Tk .Dflag=1) Then Tk .class=3
10. if(Tk .Eflag== Tk .Dflag=0) Then Tk .class=4
11. Schedule all task of class 1 parallely to all cores
12. Schedule all task of class 2 parallely to all cores after class 1 scheduled
13. Repeat step 12 for remaining classes.

IV. PERFORMANCE EVALUATION

Performance of proposed algorithm is analyzed with IRMS algorithm on the parameters of average waiting time and average turnaround time and bandwidth utilization.

Waiting time: Amount of time task spent in ready queue, waiting for the resources allocation, I/O manipulations and intercommunication with other tasks. Average waiting time indicates the average waiting time for each task needs to wait for allocation to processing unit i.e. is core.

Turnaround time: Turnaround time is the time gap between ready time to completion time. Average turnaround time indicates average time each task needs to get completely executed after it is allocated to processing unit

Utilization:
CPU utilization refers to a computer's usage of processing resources, or the amount of work handled by a CPU
In order to evaluate the performance of proposed scheduling strategy, simulation is done on three case studies.

Case Study 1:

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution time</th>
<th>Deadline</th>
<th>Probability</th>
<th>Performance rate</th>
<th>Expected performance</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>4</td>
<td>10</td>
<td>0.3</td>
<td>0.2</td>
<td>0.7</td>
<td>0.8</td>
</tr>
<tr>
<td>T2</td>
<td>3</td>
<td>8</td>
<td>0.4</td>
<td>0.3</td>
<td>0.6</td>
<td>0.7</td>
</tr>
<tr>
<td>T3</td>
<td>2</td>
<td>9</td>
<td>0.2</td>
<td>0.1</td>
<td>0.8</td>
<td>0.9</td>
</tr>
<tr>
<td>T4</td>
<td>4</td>
<td>8</td>
<td>0.4</td>
<td>0.3</td>
<td>0.6</td>
<td>0.7</td>
</tr>
</tbody>
</table>

In case study 1 there are four tasks to schedule. By using given deadline, execution time and failure probability performance is calculated. Here there are three instances of failure probability considered. And according to performance calculate the overall performance of the all the tasks. Now applied proposed algorithm, by using algorithm divide tasks into four different classes and according to classes set the priorities as shown in above case study. Now assign the tasks according to priorities of tasks.
Figure 4.1: Task Allocation for Case Study 1

Fig. 4.1 shows task allocation for Case Study 1 which have task set comprising four tasks. Each task is allocated to processing unit according to scheduling strategy. Task is allocated in non-preemptive manner to processing unit for the time units needed to get task executed. It can be seen that proposed algorithm respects the contribution of each task as well as deadline for each task.

Case Study 2:

Table 4.2: Task set\textsubscript{2} with performance and priority

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution time</th>
<th>Deadline</th>
<th>Probability</th>
<th>Performance rate</th>
<th>Expected performance</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>2</td>
<td>12</td>
<td>0.2</td>
<td>0.1</td>
<td>0.8</td>
<td>0.9</td>
</tr>
<tr>
<td>T2</td>
<td>3</td>
<td>10</td>
<td>0.4</td>
<td>0.2</td>
<td>0.6</td>
<td>0.8</td>
</tr>
<tr>
<td>T3</td>
<td>2</td>
<td>8</td>
<td>0.1</td>
<td>0.2</td>
<td>0.9</td>
<td>0.8</td>
</tr>
<tr>
<td>T4</td>
<td>4</td>
<td>10</td>
<td>0.3</td>
<td>0.4</td>
<td>0.7</td>
<td>0.6</td>
</tr>
<tr>
<td>T5</td>
<td>4</td>
<td>12</td>
<td>0.1</td>
<td>0.2</td>
<td>0.9</td>
<td>0.8</td>
</tr>
<tr>
<td>T6</td>
<td>3</td>
<td>9</td>
<td>0.2</td>
<td>0.3</td>
<td>0.8</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Fig. 4.2 In case study 1 there are six tasks to schedule. By using given deadline, execution time and failure probability performance is calculated. Here there are three instances of failure probability considered. And according to performance calculate the overall performance of the all the tasks. Now applied proposed algorithm, by using algorithm divide tasks into four different classes and according to classes set the priorities as shown in above case study. Now assign the tasks according to priorities of tasks.
Fig. 4.2 shows task allocation for Case Study 2 which utilizes all available cores up to some extent. Although in this case core2, core4 are not utilized to their fullest as there is no ready task at certain time due to performance of task. On the other hand it can also be observed that concurrency is achieved while executing tasks due to use of multi-core processing platform.

Case Study 3:

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution time</th>
<th>Deadline</th>
<th>Probability</th>
<th>Performance rate</th>
<th>Expected performance</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>3</td>
<td>12</td>
<td>0.4</td>
<td>0.6</td>
<td>0.54</td>
<td>4</td>
</tr>
<tr>
<td>T2</td>
<td>1</td>
<td>7</td>
<td>0.2</td>
<td>0.8</td>
<td>0.61</td>
<td>1</td>
</tr>
<tr>
<td>T3</td>
<td>4</td>
<td>25</td>
<td>0.1</td>
<td>0.9</td>
<td>0.46</td>
<td>8</td>
</tr>
<tr>
<td>T4</td>
<td>2</td>
<td>14</td>
<td>0.3</td>
<td>0.7</td>
<td>0.6</td>
<td>6</td>
</tr>
<tr>
<td>T5</td>
<td>2</td>
<td>9</td>
<td>0.2</td>
<td>0.8</td>
<td>0.6</td>
<td>2</td>
</tr>
<tr>
<td>T6</td>
<td>3</td>
<td>18</td>
<td>0.1</td>
<td>0.9</td>
<td>0.46</td>
<td>7</td>
</tr>
<tr>
<td>T7</td>
<td>5</td>
<td>12</td>
<td>0.2</td>
<td>0.8</td>
<td>0.49</td>
<td>5</td>
</tr>
<tr>
<td>T8</td>
<td>1</td>
<td>10</td>
<td>0.1</td>
<td>0.9</td>
<td>0.54</td>
<td>3</td>
</tr>
</tbody>
</table>

In case study 3 there are eight tasks to schedule. By using given deadline, execution time and failure probability performance is calculated. Here there are three instances of failure probability considered. And according to performance calculate the overall performance of the all the tasks. Now applied proposed algorithm, by using algorithm divide tasks into four different classes and according to classes set the priorities as shown in above case study. Now assign the tasks according to priorities of tasks.
Fig. 4.3 shows allocation of tasks from task set \( t_3 \) to processing units for their execution in non-preemptive manner. All cores are utilized; therefore concurrency in task execution is achieved.

V. PERFORMANCE ANALYSIS

Minimum waiting time for task ensures that it is allocated to processing unit in timely manner. Therefore waiting time for task should be as less as possible.

Fig. 5.1 depicts the average waiting time comparison between IRMS algorithm and proposed algorithm. It can be observed that average waiting time is significantly less for proposed algorithm compared to IRMS algorithm.
Fig 5.2: Average Turnaround Time Comparison

Fig 5.2 depicts average turnaround time comparison between IRMS algorithm and proposed algorithm. It can be observed that average turnaround time is less compared to IRMS algorithm.

Fig 5.3: Bandwidth Utilization Analysis

In proposed scheduling algorithm for executing application on cores average bandwidth utilization increases as number of tasks get increased for execution.

VI. CONCLUSION

Problem of contribution of tasks can be solved more effectively with this approach. This dissertation proposes new scheduling policy which addresses performance of the task scheduling considering deadline and performance contribution of individual task into output of the system on multi-core processing platform. Scheduling policy is simulated for homogeneous quad core processing platform.
Results are compared with Improved RMS algorithm for uniprocessor. Results on the basis of case studies performed shows 14% decreased in average waiting time and 15.85% improvement in turnaround time and CPU utilization is greater as compared.

VII. FUTURE SCOPE

Proposed algorithm can be implementing further for aperiodic tasks in the real time scheduling, if the aperiodic task coming in the process of scheduling. we can be implement it for dependent tasks if the dependent tasks is for execution. This scheduling policy can be use on more than four cores. Performance will certainly vary with increase in number of cores in future as more number of tasks can be scheduled to be executed concurrently.

REFERENCES