Self-configurable Processor Schedule Window Algorithm

A. Prasanth Rao[1], A. Govardhan[2], and P.V.V. Prasad Rao[3]

[1] Reasearch Scholar, Dept., of Computer Science and Engineering, JNTUH, Hyderabad
[2] Professor, Dept., of Computer Science and Engineering, JNTUH, Hyderabad
[3] Program Manager, SETU Software Systems Pvt.. Ltd., IIIT Gouchibowli, Hyderabad
adirajupppy@yahoo.com, govardhan_cse@yahoo.co.in,
prasad@setusoftware.com

Abstract. Most periodic tasks are assigned to processors using partition scheduling policy after checking feasibility conditions. A new approach is proposed for scheduling aperiodic tasks with periodic task system on multiprocessor system with less number of preemptions. Our approach is self-configurable and adjusts the periodic tasks to the processor such that different types of tasks are scheduled without violating deadline constraints. The new approach proves that when all different types of tasks are scheduled, it leads to better performance.

Keywords: Scheduling, feasibility, multiprocessor, deadline, synchronous, free slots, sporadic task system.

1 Introduction

Real-Time systems are specifically designed for situations where the correctness of an operation depends not only upon its logical correctness, but also upon the time at which it is performed. Generally real time applications are event driven and the task should complete its execution within the deadline and so it should be completely determinable. Events can be classified according to their arrival pattern. In this context, events can be periodic if their arrival time is a constant or aperiodic when it is not. A task set is said to be synchronous if all offsets are equal to zero and the deadline of the task is equal to or less than its period. The polynomial test has been proposed by Baruah et al [3]. If the deadlines are equal to period, a simple polynomial test has been proposed by Liu and Layland in their seminal work [1]. A task set is asynchronous if the task arrival time is not known in advance and for each asynchronous set a synchronous set is identified. A task set can be categorized as having implicit deadline, constraint deadline or arbitrary deadlines. An asynchronous task is one which has an arbitrary deadline and should be modeled simply as a synchronous set [13].

A task set is said to be have an implicit deadline if $\forall i$, $task_i$, $d_i=p_i$: For constraints deadline the task set has $\forall i$, $task_i$, $d_i \leq p_i$**.** Finally for a task set with arbitrary deadlines no such relation stated. However, in general there are other types of tasks whose arrival times are not known in advance and these tasks are scheduled together with the periodic tasks set.

N. Meghanathan et al. (Eds.): CCSIT 2012, Part III, LNICST 86, pp. 411–422, 2012.

There are many partitioned scheduling algorithm[2,3,4,5,6,7] which is used to schedule periodic tasks [8,9]. However using partitioned scheduling algorithm, tasks are allocated to processors and each processor is allocated certain fixed number of execution units. The main disadvantage of partitioned scheduling algorithm is that the processor is not fully utilized. This means that there are certain execution time units available and these units are fragmented in the processor. To overcome the disadvantage of partitioned scheduling algorithm, we have to make use of unused cycles properly. So other types of tasks such aperiodic tasks, constrained deadline or arbitrary deadline are scheduled with periodic task sets which improve the overall performance of the system.

There are certain free slots available at different intervals of time and any one of the free slots may accommodate only a few execution time units. The individual fragments may not be large enough, so we can combine all these small fragments to execute a much bigger task. The size of these free slots is configurable with a parameter, which controls the free cycles availability and allocation of dynamic tasks to individual processor.

The main objective of this chapter to develop strategies to schedule aperiodic tasks with periodic tasks and the same model can be extended to schedule a sporadic task system.

This paper is organized as follows. Section 2 describes Theoretical Concepts Configuration Parameter presented in Section 3. Section 4 describes Reserving space for newly created tasks Sections 5 demonstrate Results and Discussions. Finally, the Conclusion and future scope are given in Section 6**.**

2 Theoretical Concepts

A periodically sampled control system which is modeled on time triggered approach. Time-Triggered tasks (τ) are characterized by a quadruple ($\emptyset$*, p, e, d*). The periodic task system involves execution of independent task system $\Gamma = \{\tau_1, \tau_2, \tau_3, \tau_4 \ldots \tau_n\}$, where each task $\tau_j \in \Gamma$. The period task generates a sequence of jobs at each integral multiple of period p_i. Each job must execute in at most e_i execution units of time and should complete before its relative deadline d_i(equals to period of task). The first job of the given task is released at phase $\emptyset$ (offset). Since periodic task system generates an infinite sequence of jobs with the k^{th} job arriving at an instance $\emptyset_i + (k\text{-}1)\, p_i \ \forall i=1, 2, 3, \ldots, k$ and each job should complete before $\emptyset_i + (k)\, p_i$. Before presenting finding free-slots algorithm we need to define Basic Terminology.

2.1 Basic Terminology

In this section we look at definitions which help us to understand the partition parameter and using this parameter we can dynamically configure processor window.

Definition 2.1 (Total Execution Period P_{max}). The total execution time units of a given task system Γ^1 is equal to or less than total execution period (P_{max}), then the task system is feasible on uniprocessor system.

$$P_{\max} = \max(p_1, p_2, p_3, \ldots\ldots\ldots\ldots, pi, \ldots\ldots p_n) \quad (2.1)$$

Definition 2.2 (Maximum Execution Units e_{max}). e_{max} is maximum execution units which is defined among a set of n tasks.

$$e_{max} = \max(e_1, e_2, e_3, \ldots, e_m) \quad (2.2)$$

Definition 2.3. (Configuration Parameter δ). The configuration parameter δ divides the time axis into two windows. One window is used to schedule periodic task sets and the other window is used to schedule aperiodic task sets.

Definition 2.4 (Scheduling Condition). The general scheduling condition given RMCT [2]

$$P_{max} \geq \lfloor P_{max} / p_1 \rfloor e_1 + e_{max(P)} \quad (2.3)$$

Using the above definitions we can understand partition condition which divides processor window into two parts. Before defining partition condition we need to explain about task execution modeling in next section.

2.2 Task Execution Modeling

A scheduling algorithm provides a set of rules that determine the processor(s) to be used and tasks to be executed at any particular point of time. There many scheduling algorithm were presented in the literature [7,10,11] and also scheduling heuristics [8,13] developed. These entire algorithms are allocating tasks to the processors using partition scheduling policy which results some unused free slots on each processor. Also we configure these unused free slots to schedule different type of tasks together. For further improvement in resource utilization if we make use of these free slots to schedule dynamic tasks.

2.3 Scheduling Conditions

The schedulable condition for task sets scheduled under RM is based on utilization of a processor and period oriented. All scheduling conditions which were mentioned above are oriented towards utilizations i.e. the relative value of task utilization was taken into account. The performance of these algorithms is limited because they fail to consider the relative values of task periods. There are many period oriented scheduling algorithms were developed RMGT [8] RMCT [2] and utilization based algorithms [3][8].

The Rate Monotonic Critical Tasks (RMCT) algorithm is developed based on the maximum execution period (P_{max}) and assumes all tasks in a queue are arranged with decreasing period. The total execution units (T) of all incoming periodic tasks can be computed and the RMCT may allocate maximum possible tasks to given processor till condition 2.4 satisfied. The RMCT Algorithm [2] identifies the number of processors and also total execution time units (T) given to each processor. The δ in the if loop known be configuration parameter.

If ($T>\delta P_{max}$) then
{ increment processor index. (2.4)
else
{ allocates to same processor.
}

In the following section we show how this condition in 2.4 [2] helps to regulate the task being scheduled.

3 Configuration Parameter

Let us consider task system Γ^1 {τ_1, τ_2, τ_3 } and τ_1=(5,2), τ_2=(7,1), τ_3=(10,4). Apply RMCT [2] algorithm, p_{max}=10, T=10 units, so all tasks are allocated to only one processor. In this situation δ=1 and processor fully loaded with periodic set as shown Figure 1.From figure we conclude that the processor fully loaded and there will be no free slots available and these type of tasks should meet deadline.

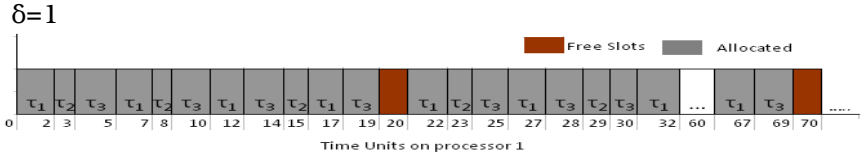

Fig. 1. Task allocation to processor 1

However when we change δ=0.8, we require two processor to schedule above tasks system as shown in Figure 2. From the figure we conclude that two processor required to schedule above task system and each processor some free slots are available.

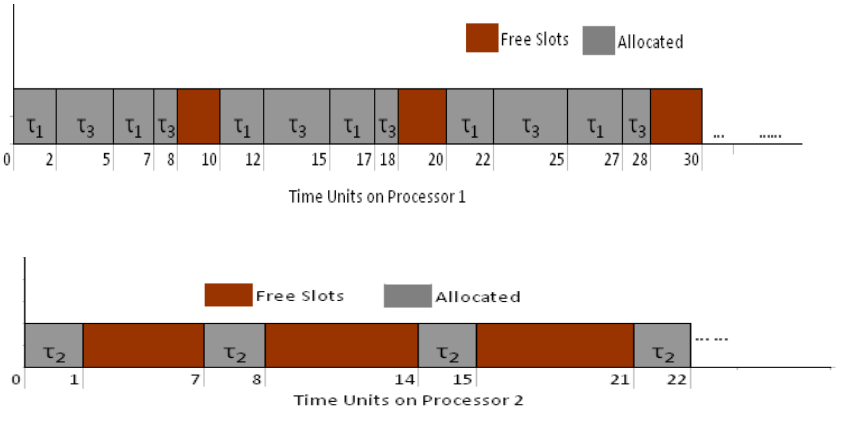

Fig. 2. Tasks allocation two processor system

However we can further increase these free slots by decreasing value of δ. The δ=0.4 as shown in Figure 3.

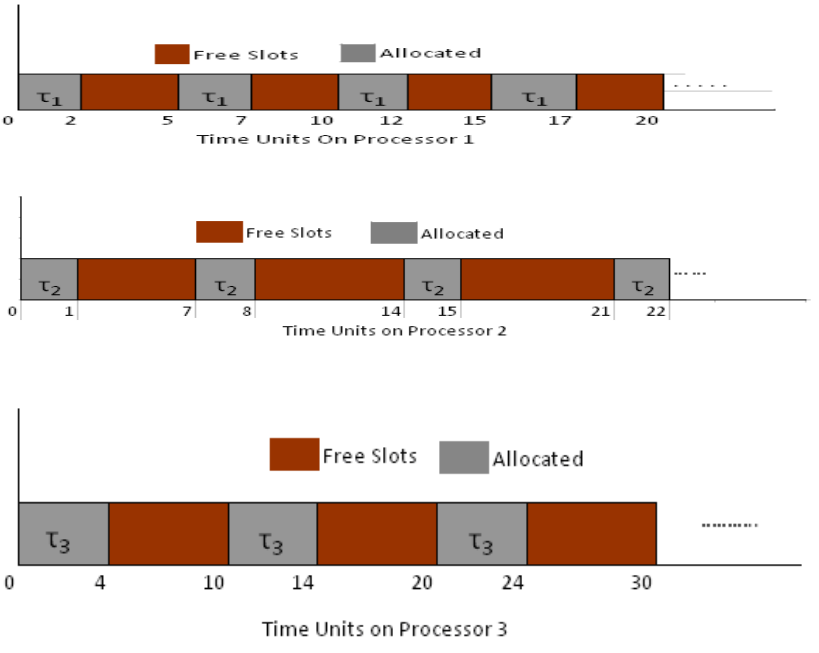

Fig. 3. Task allocation three processor system

In this situation, minimum of three processor required and each processor can allocate one periodic tasks. However it may increase number of processor but there will be more number of free slots are available. This helps us to schedule different type of tasks together and we mix up both high priority and low priority tasks. Not only that an aperiodic tasks whose generation not known in advance can be scheduled with periodic tasks. The configuration parameter dynamically can change depending upon the application. So in order to regulate the proportion of aperiodic tasks we consider a configuration parameter δ and must be selected such that both periodic and aperiodic tasks can be scheduled together. δ is chosen such that at least one periodic task is allocated to an individual processor. δ, the configuration parameter divides the available window into parts ,where one part is reserved for fixed priority algorithm [4]and other part is dynamic priority algorithm[10]. We observe that for further decreases of δ value, there will be at least one periodic task cannot be allocated any processing element (theorem 4.1).

By choosing an appropriate value of δ, we can use our scheduling algorithm in two different ways. Firstly, it can be used to schedule the task set on an individual processor after checking feasibility analysis. Secondly, it can be used to find free slots on each processor and in turn these are used for allocation of dynamically created tasks. This is called as the Partition Condition which is discussed in the next section.

3.1 Partition Condition

The partition condition divides window into two parts where one window can be used for scheduling periodic tasks and other window is to schedule aperiodic tasks. We propose here the theorem 3.1 states that the window is divided such that at least one periodic task is allocated to each processor.

Theorem 3.1. For given periodic task system Γ^1 the partition condition states that each processor allocates at least one periodic task using configuration parameter δ and its value lies between $[\delta_{min,}\ 1]$.
Where $\delta_{min,}=e_{max}/p_{max}$

Proof: Let $\{\tau_1, \tau_2, \tau_3, \tau_4 \ldots \tau_n\}$ be n tasks and all tasks are arranged with increasing priorities and first in queue will be given least priority.

Let τ_n, τ_k is two tasks whose maximum periodicity (Def.3.2) and whose maximum execution time (Def.3.7) are (p_{max}) and (e_{max}) respectively.

If $e_i<p_i$ then $task_i$ is schedulable on processor j otherwise the task is infeasible. This implies that there will be a task whose maximum execution units are e_{max} and all other tasks are in a queue below this value. For any given task τ_k i.e. $p_k<p_{max}$ and that implies $e_{max}<p_{max}$. This means minimum δ_{min} equals to e_{max}/p_{max} and its maximum value equals to 1.The configuration parameter value always lies in between $[\delta_{min,}\ 1]$. So the minimum execution units allocated to each processor equals to e_{max} such that selecting δ value in the range $[\delta_{min,}\ 1]$. There will be no task allocated to processor if $\delta< \delta_{min}$ and so at least one periodic task allocated to processor if δ lies in the range $[\delta_{min,}\ 1]$.

Hence Proved

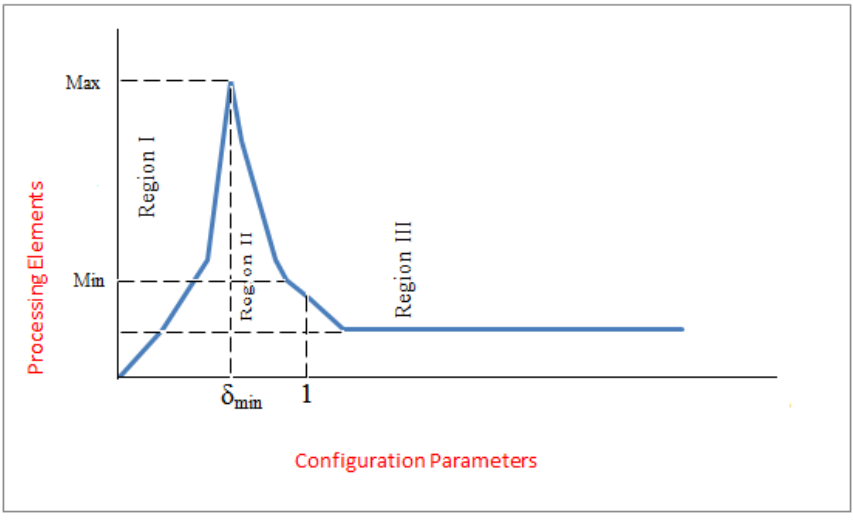

Fig. 4. Feasible Region Graph

The Figure 4 drawn between configuration parameter and processing elements required for computing given load. The observations of three regions from Feasible Region Graph are listed below.

Region I: Configuration parameter value in between [0, δ_{min}). Task allocation cannot done properly and some tasks are unassigned to processing elements. When δ =0, none of the task assigned.

Region II: Configuration parameter value in between [δ_{min},1]. As we move δ_{min} to 1,the number of processing elements required to compute for given task system get decreases. However, more number of free slots available on each processor at δ_{min} and deceases further. In this region all tasks meet deadline and RM schedulable conditions.

Region III: The parameter δ increases further after reaching its value 1, tasks are assigned with lesser number of processing elements. However, they will not meet deadline and also not RM schedulable.

Region II best suitable for scheduling periodic tasks and adjusting parameter δ, we schedule different types of tasks are together.
Depending upon the type of an application we can set the value of δ and based on this value the processor allocates fixed number of periodic tasks and also has few free slots to accommodate aperiodic tasks. The same model can be extended to schedule sporadic task system [13].

When new task arrives at processor m_i , at phase Ø with deadline d_i and execution time e_i it is scheduled between the time interval Ø and Ø+d_i. When the new task arrives the algorithm immediately searches for a free slot to schedule the task locally otherwise it is sent to the group scheduler. Few processors are grouped together and allocation of tasks among these processors can be monitored by group scheduler. An integrated procedure is desirable to schedule different types of periodic/aperiodic/sporadic task system**.** Before presenting an integrated approach we need to find free-slot in given interval and discussed in the next section.

3.2 Availability of Slots at Fixed Interval

Initially, the centralized scheduler allocates a fixed number of execution time units to an individual processor and in each processor available free slots are computed. The total number of processing elements m will be divided into a number of small groups. Each group maintains a group scheduler which contains information about all processing elements within it. The Algorithm 3.1 is used to compute free slots and this information is available at group scheduler. A group scheduler maintains a table which contains information about each processor-ID, total-execution units, planning cycle (M) and size of fixed free slots.

Let the number of tasks allocated by RMCT algorithm j = n_j.
Planning cycle or LCM units of periodic tasks allocated to processor j=M_j
The number of occurrences l = M_j/phase

Initialize first task in priority queue and fixed slots in the interval given by

$$\left(k.phase_i, K.phase_i + e_{ij}\right), \forall k = 0,.....,1 \tag{3.1}$$

The rest of tasks which are present in the queue verify availability of free slots and each task from queue is allocated free slots using algorithm 3.1. The fixed allocated slots means that tasks which are assigned to particular processor execute only mentioned slots. The condition which is used in RMCT is said to necessary and sufficient i.e. property of RM algorithm is satisfied. The allocation of fixed time slots means that the execution of the task is predefined in allocated time frame. This results in a very low number of preemptions.

Algorithm 3. 1. Finding Free Slots

```
I for(j=0;j≤m;j++)
Read number of tasks (n_ij ,M,Phase);
for i=0;
//First task only.
Occurrences l = M/phase
for (k=0;k≤l; l++)
Slots fixed for task_i= [k*phase_i, K*phase_i + e_ij]
Available slots =[ K*phase_i + e_ij, (K+1)phase)
II for (i=1;i≤n_ij ;i++)
{
Check in available slots.
Occurrences l=M/phase_i
for (k=0; k<=l;k++)
  {
  Pick up one by one available slot
  If (upperbound-lowerbound>=eij)
  Slots fixed = [lower bound, lowerbound+eij]
  Available slot =[lowerbound+eij, upper bound]
  else
  Keep available slot as it is.
  }
  }
```

4 Reserving Space for Newly Created Tasks

On processor p, identify free slots available in the interval Ø and Ø+ d_i. If any one of the free slots are sufficient to accommodate newly arrived task then allocate that task to processor otherwise it searches another processor in group scheduler. If one processor not sufficient then task may split into two fragments.

Let the space reserved on processor m_i for one portion of the split-task be $x[m_i]$ and the space reserved for second portion of the split-task on processing element (m_j) be $z[m_j]$.Likewise all the parts of the split-task reserve spaces on processors which are

within the group. The dispatching is simple. If processor m_i reserves at time t and other portion of the split task assigned another processor m_j reserve after time t + x $[m_i]$.

Assume S_1, $S_2 \ldots S_n$ are the sizes of free slots available on the processor in the interval (0, M].The free slot which has the maximum size, has to be identified and has to be denoted by S_{max}, then splitting of task can be done.

S_{max} = maximum slot size within the interval Ø and Ø+ d_i.
S_{max} = max (S_1, S_2,..S_k), where k free slots are available in mentioned Interval.

The space reserved for one portion of task should be equal to S_{max} i.e. $x[m_i]= S_{max}$

Next a suitable processor m_j should be searched for the remaining portion of split-task such that $Z[m_j] = e_{ij} - S_{max}$

The number of processing elements in given system will be m and this number divided smaller groups. Each group contains smaller number of processing elements in order to reduce communication latency. The next section presents an integrated approach to schedule different type of tasks.

4.1 Integrated Procedure to Schedule Different Tasks

The integrated approach integrates all three schedulers (local, global and group) to provide a complete solution to schedule real tasks among different processing elements. This scheme has all the three components and also takes interactions among different processing elements. Each processor has a local scheduler and is allocated a fixed number of execution time units in a given planning cycle. As we know, there will certain free slots available and these intervals were fixed.

The algorithm 4.2 locks those free slots in the given group of processing elements which one optimal by adequate for given tasks taking its phase, periodicity, execution time, deadline. When a dynamic task arrives at time t, the task tries to schedule locally otherwise it searches free slots in that particular processor group. The algorithm 4.1 is used for getting number of free slots in each processor and size of each free slot. The size_free_slots [ ] gives us the size of free slot and no_free_slots [req_size] gives us number of free slots in that particular interval for each and every processing element. The algorithm 4.2 is used for finding optimal size of free slots in the interval θ and θ + d.

Algorithm 4.1. Getting free slots in the interval t and t+d in given group

```
size_free_slots [req_size]
no_free_slots [req_size]
Locked [req_size]
for (i=0;i<req_size; i++)
{
  No_free_slots[i]=available_fslots(i,t,d);
  size_free_slots[i]=available_fssize(i,t,d)
}
```

```
Available_fslots (int i, int t, int d)
{
  get sizes of free slots in the interval t and t+d;
}
```

Algorithm 4.2. Finding optimal free-slot in the interval t and t+d for newly arrived task

```
  //optimal allocation:
   optimal_fs_size=size_free_slots
   for (int i=0;i<req_size; i++)
   {
  //pick one value greater than min_req and less than
  remaining all
      If(size_freeslots[i]•min_req&& locked[i]=0)
      {
        If (optimal_fs_size•size-freeslots[i])
          optima_fs_size = size_freeslots[i];
       }
   }
  If (optimal_fs_size<min_req)
   {
     //assign nearest fsize to splitting task
     //find out maximum value so that will get the
  nearest value
     Optimal_fssize=Max(availability_fslot[]);
    //lock that processing element;
    Locked[j]=1;
    min_req=min-req-optimal_fssize;
    //repeat optimal allocation block for remaining
   }
   else
   {
     //no need to split task directly lock that
  processing element;
     Locked[i]=1;
     Lock-index[j] with request processor[i]
   }
```

In order to use our algorithm, we need to ensure that each task is processed on only one processor at any point of time. Task splitting must therefore address three important challenges (i) Dispatching algorithm to be developed for ensuring that two pieces of a task do not execute simultaneously (ii) Design a schedulable test for the dispatching algorithm. (iii) Order of execution of two pieces is maintained properly.

5 Results and Discussions

The m processor system divides into smaller number of groups and splitting of task can be made within the group. However, we illustrate our results with one example as shown in Figure 3.When δ=.8,The task system requires three processor system and all these processors are grouped together and free slots in each processor as shown table 5.1. Whenever an two aperiodic task1 (4, 4, 6) and aperiodic task2 (5, 5, 7).System invokes finding free slots and also calls finding optimal slots among available free slots in the interval θ and θ + d.

Table 1. Free slots in each processor for given group

Processor-ID	Allocated Blocks	Free Slots	Locked slots
1	(0,2),(5,7),(10,12)…	(2,5),(7,10) (12,15)…..	
2	(0, 1),(7,8),(14,15)…	(1,7),(8,14), (15, 21)….	(5,7),(8,11)
3	(0,4),(10,14),(20,24)…	(4,10),(14,20), (24, 30)……	(4,8)

For first aperiodic task when free slots algorithm invoked in the interval θ and θ + d and it finds suitable slots are (4,10) on processor 3,(4,5),(7,10) on processor 1 and (4,7),(8,10) on processor 2.However (4,8) slot is more suitable and this slot is locked for it. Similarly, the other aperiodic task arrives at phase 5 and again it searches suitable slots in given group. So aperiodic task1 allocated to processor 3 and aperiodic task2 allocated processor 2.We have shown only how our algorithm works with simple example. Simulation works are under progress.

6 Conclusions

This paper provides a solution to make use of unused free-slots on existing processors. Different types of tasks are scheduled with periodic task sets. Initially, the system tries to schedule the newly created dynamic tasks to one of the available processors. However, there are a few cases where CPU cycles are not sufficient to execute a given task. In such scenarios, task splitting can take place between two processors thus improving resource utilization. As a future enhancement, we will group the processing elements using a Maekawa set which reduces communication delay and there is a need develop an integrated scheduler (i.e. local scheduler, group scheduler and centralized scheduler) for real-time task systems.

References

1. Liu, C., LayLand, J.: Scheduling Algorithms for Multiprogramming in Hard Real–Time Environment. JACM 10(1), 174–189 (1973)
2. Prashanth Rao, A., Govardhan, A.: An Improved Period Oriented Scheduling Algorithm for Real Time Systems. Ijesce Research Science Press 3(1) (January-June 2011)
3. Cheng, S., Stankovic, J.A., Ramamritham, K.: Scheduling Algorithm for Hard Real Time Systems: A Brief Survey. In: Tutorial: Hard Real Time Systems, pp. 150–173. EFF Press (1988)
4. Gafford, J.D.: Rate Monotonic Scheduling. IEE Micro, 34–39 (June 1991); Liu, J.W.S.: Real Time Systems, IInd edn. Pearson Education (1991)
5. Johnson, D.S.: Near Optimal Bin Packing Algorithms, PhD Thesis. MIT (1973)
6. Buttazzo, G.C.: Hard Real-Time Computing Systems Predictable Scheduling Algorithms and Applications. Kluwer Academic Publishers (1997)
7. Joseph, M. (ed.): Real-time Systems Specification, Verification and Analysis. Tata Research Development & Design Centre (June 2001)
8. Burchard, A., Liebeherr, J., Oh, Y., Son, S.H.: New Strategies for Assigning Real-Time Tasks to Multimocessor Systems. IEEE Transactions on Computers 44(12) (December 1995)
9. Lauzac, S., Melhem, R., Mossé, D.: An Improved Rate-Monotonic Admission Control and Its Applications. IEEE Transactions on Computers 52(3) (March 2003)
10. Siva Ram Murthy, C., Manimaran, G.: Resource Management in Real-Time Systems and Networks. PHI Learning Private Limited (2009)
11. Zmaranda, D., Gabor, G., Popescu, D.E., Vancea, C., Vancea, F.: Using Fixed Priority Pre-emptive Scheduling in Real-Time Systems. International Journal of Computers, Communications & Control VI(1), 187–195 (2011)
12. Pellizzoni, R., Lipari, G.: Feasibility Analysis of Real-Time Periodic Tasks with Offsets. Real-Time Systems 30(1-2) (May 2005)
13. Anderson, B., Bletsis, K., Baruah, S.: Arbitrary-Deadline Scheduling Sporadic Tasks on Multiprocessor, Technical Report HURRAY-TR-080501