SCHEDULING DIFFERENT CUSTOMER ACTIVITIES WITH SENSING DEVICE

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

[1]Reasearch Scholar, Dept., of Computer Science and Engineering, JNTU, Hyd

adirajupppy@yahoo.com

[2]Professor, Dept., of Computer Science and Engineering, JNTU, Hyd

govardhan_cse@yahoo.co.in

[3]Program Manager, SETU Software Systems Pvt..Ltd., IIIT Gouchibowli, Hyd

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 different activities with one periodic task within the system. In this paper, control strategies are identified for allocating different types of tasks (activities) to individual computing elements like Smartphone or microphones. In our simulation model, each periodic task generates an aperiodic tasks are taken into consideration. Different sets of periodic tasks and aperiodic tasks are scheduled together. This new approach proves that when all different activities are scheduled with one periodic tasks leads to better performance.

KEYWORDS

Scheduling, feasibility, multiprocessor, deadline, synchronous, free slots, activities, configuration parameter.

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. An asynchronous task is one which has an arbitrary deadline and should be modeled simply as a synchronous set [13].

There are many partitioned scheduling algorithm[1,2,3,4,5,6,7] which is used to schedule periodic tasks [8,9]. However using partitioned scheduling algorithm, tasks are allocated to processors and

DOI : 10.5121/ijait.2012.2205

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, different customer activities such loading a file, payments,etc. are scheduled with periodic task sets which improve the overall performance of the system. The excess unused computing cycles (free slots) on each computing device can be computed and these computing cycles are used to schedule different tasks.

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 different customer activities can be monitored with one periodic task.

Motivation: Modern mobile devices are packed with lots of computing power with latest processors and highly energy efficient hardware to offer longer battery life. It becomes easy to harness that cumulative processing power to process bigger and highly computing intensive tasks using mobile device cloud. By using mobile server cloud computing technologies, this power can be harnessed in the network. The application like shopping mall traffic monitoring system. The practical scenario of a customer who goes to a shopping mall with a mobile phone handset. The communication system is the mall immediately senses the presence of the mobile device and sends out a request to this customer for permission to add this device to its network. As soon as the customer accepts this request, his phone gets added to the network and it starts getting used to monitor and regulate the customer's business activities like identification and procurement of times, bill payments etc. Similar systems can also be used to regulate traffic, identify hot spots and implement effective crowd control mechanisms. All of these items require mobile devices to collaborate to increase computation power in order to perform real-time analysis of sensed data.

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 Integrated Procedure to Schedule Different Tasks. Finally, the Performance Results 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 $\tau_j \in \Gamma$.

2.1. Assumption about Task System

By analyzing above applications mentioned in the motivation, the assumption of system model can be presented below

Periodic Task Model

The periodic 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 sensing data from different devices are known be periodic and assumptions are listed below.

a. The tasks which are generated from different sensors are known to be the periodic tasks. The different instances (jobs) of the periodic tasks are generated at regular interval of time.
b. All periodic tasks are independent and there is no dependence among these tasks.
c. These types of tasks may generate another type of tasks known to be aperioidc or activities.

Before presenting finding free-slots algorithm we need to define Basic Terminology.

2.2. 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.

$$P_{\max} = \max(p_1, p_2, p_3, ..., pi, ...p_n) \quad \ldots\ldots(1)$$

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

$$e_{max} = \max(e_1, e_2, e_3, ..., e_m) \quad \ldots\ldots(2)$$

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]

$$P_{max} \geq \lfloor P_{max} / p_1 \rfloor e_1 + e_{max(P)} \quad \ldots\ldots(3)$$

Definition 2.5: (Cut-Off Configuration Value). $[\delta_{cut\text{-}off}\ 1]$ is known to be feasible region and if $\delta < \delta_{cut\text{-}off}$ then atleast one periodic task cannot be allocated

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.

2.3. Task Execution Modelling

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.

2.4. 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 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. ….(4)
else
{Allocates to same processor
}

In the following section we show how this condition in 4 [2] helps to regulate the task being scheduled.

3. CONFIGURATION PARAMETER

Let us consider task system Γ^1 $\{\tau_1, \tau_2, \tau_3\}$ and $\tau_1=(5,2)$, $\tau_2=(7,1)$, $\tau_3=(10,4)$. Apply RMCT [2] algorithm, $p_{max}=10$, T=10 units, so all tasks are allocated to only one processor. In this situation δ=1 , when 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.

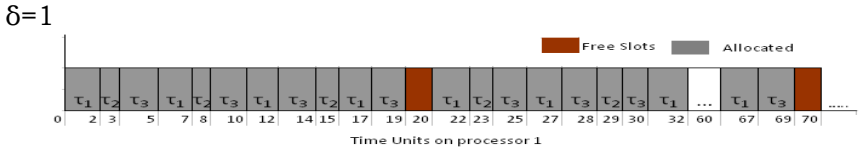

Figure 1. Task allocation to processor 1

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 processors required to schedule above task system and each processor some free slots are available.

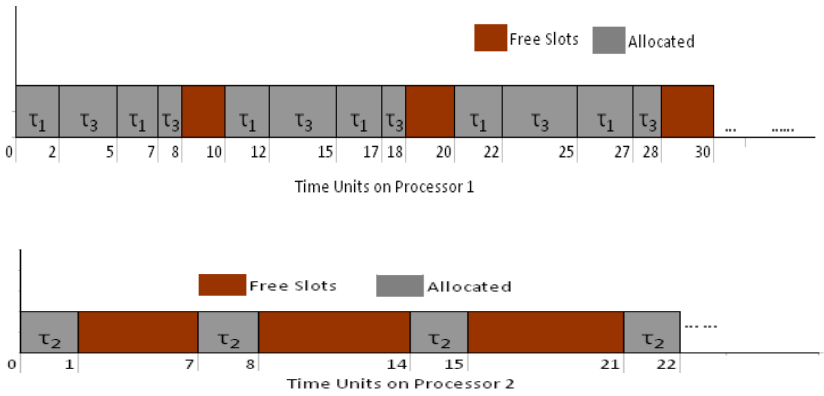

Figure 2. Tasks allocation two processor system

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

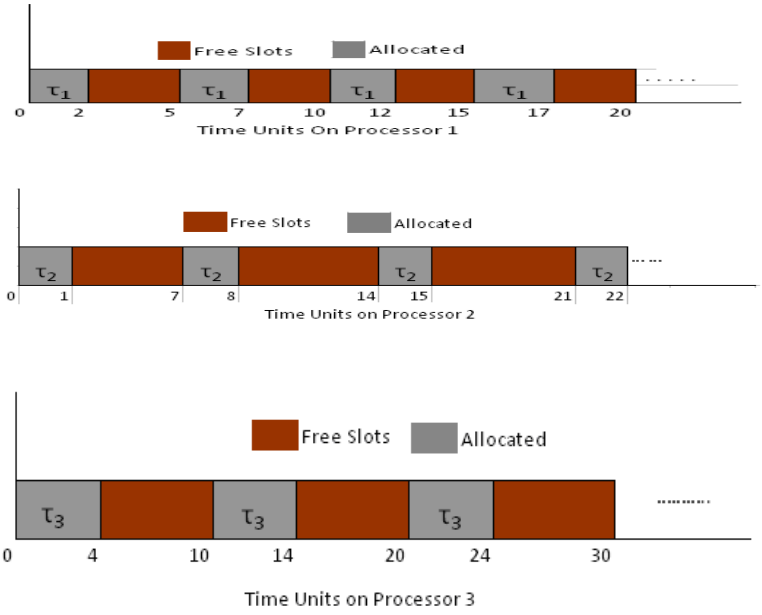

Figure 3. Task allocation three processor system

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 is 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 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. The theorem 1 is proposed which states that the window is divided in such a way that at least one periodic task is allocated to each processor.

Theorem 1: Given periodic task system $\Gamma^1\{\tau_1, \tau_2, \tau_3, \tau_4 \ldots \tau_n\}$, the partition condition states that each processor allocates at least one periodic task using configuration parameter δ and its value lies between $[\delta_{cut\text{-}off}, 1]$. Where $\delta_{cut\text{-}off}=e_{max}/p_{max}$

Proof: Let $\Gamma^1\{\tau_1, \tau_2, \tau_3, \tau_4 \ldots \tau_n\}$ 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 be 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 not feasible. 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 ,$p_k<p_{max}$ and that implies $e_{max}<p_{max}$. This means minimum $\delta_{cut\text{-}off}$ equals to e_{max}/p_{max} and its maximum value equals 1.The configuration parameter value always lies in between $[\delta_{cut\text{-}off}, 1]$.So the minimum execution units allocated to each processor equals e_{max} so that δ value is selected in the range $[\delta_{cut\text{-}off}, 1]$. There will be no task allocated to processor if $\delta< \delta_{cut\text{-}off}$ and so at least one periodic task is allocated to processor if δ lies in the range $[\delta_{cut\text{-}off}, 1]$. **Hence Proved**

Depending upon the type of application, the value of δ can be set, based on this value the processor allocates fixed number of periodic tasks and also has few free slots to accommodate aperiodic tasks. Three different portions observed from graph are shown in Figure 4 which is presented below.

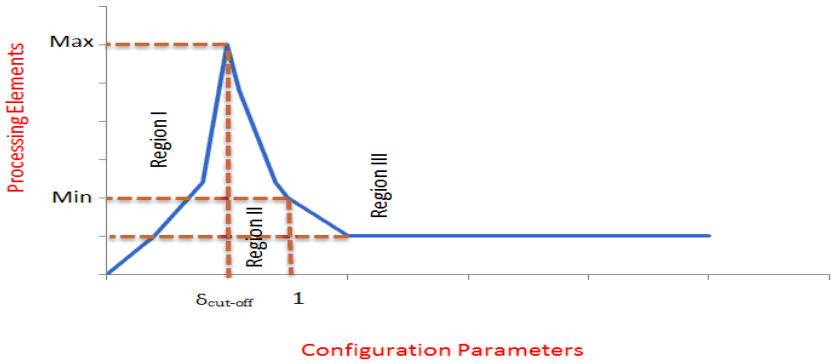

Figure 4: Feasible Region Graph

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

Region I: Configuration parameter value lies in between [0, $\delta_{cut\text{-}off}$). So task allocation cannot be done properly and some tasks are unassigned to processing elements. When δ =0, none of the tasks can be assigned.

Region II: Configuration parameter value lies in between [$\delta_{cut\text{-}off}$, 1]. When the value changes from $\delta_{cut\text{-}off}$ to 1, the number of processing elements required for computing a given task system decreases. However, maximum number of free slots are available on each processor at $\delta = \delta_{cut\text{-}off}$, when the value of δ increases further then $\delta_{cut\text{-}off}$, the number of available free slots decreases.

Region III: If the value of the parameter δ exceeds 1 then tasks are assigned with lesser number of processing elements. However, they will not meet the deadline and also they will not be RM schedulable.

Region II is best suitable for scheduling periodic tasks, adjusting parameter δ, different types of tasks are scheduled together.

The same model can be extended to schedule customer activities or sporadic task system [4]. 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. In order to allocate dynamic tasks, development of an integrated procedure to schedule different types of tasks is needed. If the available free slots are not sufficient for allocation of dynamic tasks then it becomes necessary to split them among multiple processing elements. So the next section3.2 presents conceptual ideas to develop appropriate scheduling algorithms.

3.2 Conceptual Foundations

Before presenting new algorithm RMCT-Split, one is required to understand certain concepts to design and implement an algorithm to schedule the task sets.

Task Splitting

As mentioned earlier, partitioned scheduling algorithm RMCT allocates fixed number of tasks to an individual processor and by doing this there will be some unused free time slots on each processor. This unused free space may be a very small fragment and all the available small

fragments can be combined to form a much bigger space. In order to execute a bigger task, it should be split across multiple processors. This idea has been used by several researches [2, 3, 4, 7] for scheduling sporadic tasks or periodic task with implicit deadline. In this work, the task splitting is applied to aperiodic tasks or constraint deadline or arbitrary deadline. Our work mainly focuses on the possibility of splitting newly arrived task into two or more pieces and allocating them on different processing elements. Our approach is similar to previous works but differs in the following aspects

- Task assignment and splitting of task mechanism is based on total execution units in a particular interval of time.
- The period oriented condition can be used instead of utilization based schedulable test.
- Different types of tasks are tried for scheduling on the same processor; if that is not possible, task splitting among multiple processing elements is performed.
  The Dispatching algorithm in the task model addresses two important issues
- Ensures that two pieces of the same task do not execute simultaneously on different processors.
- Schedulable test is performed between two intervals.

Fixed numbers of processors are grouped together and task splitting scheme is used among those processing elements only. This helps us to improve system throughput, response time and in reduction in latency time.

Dispatching

Different processors harmonize in time when task can be split into two pieces to execute on these processing elements. One piece of split-task is allocated to processor m_1 where reserve space is X $[m_1]$ and other piece is allocated to processor m_2 where reserve space is Z $[m_2]$.The non-split task, assigned to processor m with earliest deadline is selected for execution.

When a new task arrives at processor mi, at phase Ø with deadline d_i and execution time e_i it is scheduled between the time interval Ø and Ø+d_i. If sufficient time is available in that particular interval then the task is allocated to the same processor otherwise the task will be split into two fragments. One fragment is executed at one processor mi and other fragment is executed at another processor m_j. The splitting of the task is done in such a way that the dispatchers on each processor make sure that the fragment tasks are never executed on two processors at the same time but one particular split task is always ahead of the other split task.

In this work, task-splitting to aperiodic task with certain time constraints is applied, thus allowing an individual task to be split between the processors. The same model can be extended to schedule sporadic task system [4].

Whenever a new task arrives, the algorithm immediately searches for a free slot to schedule the task locally otherwise it is sent to the group scheduler. A few of the processors are grouped together known as group scheduler. The splitting of task can be performed within the group and it searches suitable processor for the newly arrived aperiodic or asynchronous task. An integrated procedure is desirable to schedule different types of periodic tasks, aperiodic/ asynchronous tasks**.** Before presenting an integrated approach there is a need to find free-slot in the given interval, the necessity of finding free-slots in the given interval will be discussed in the next section 4.2.3.

3.3. 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 total availability processing capacity within the group equals to sum of the free slots and can be used to execute a much bigger task. The Algorithm 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. All the pieces in a split-task should be scheduled within a particular group only.

Let the number of tasks allocated by RMCI algorithm j = n_j.
Planning cycle 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

$$(k, phase_i, K.phase_i + e_j) \forall k = 0,.....1 \quad \text{..... (5)}$$

Using above formula, fixed slots can be generated apart from available slots. The rest of tasks which are present in the queue check availability of free slots and each task from queue are allocated using Algorithm 1. Fixed allocated slots means that tasks which are assigned to particular processor execute only on mentioned slots. This condition which is used in RMCT is said to be 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 1. Finding Free Slots

```
I.  for(j=0;j≤m;j++)
      a) Read number of tasks (nij ,M,Phase);
         for i=0;
         //First task only.
         Occurrences l = M/phase
      b) for (k=0;k≤l; l++)
           Slots fixed for taski= [k*phasei, K*phasei + eij]
           Available slots =[ K*phasei + eij, (K+1)phase)
II.  for (i=1;i≤nij ;i++)
        {
           Check in available slots.
           Occurrences l=M/phasei
     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.
            }
          }
```

The group scheduler maintains a status table which contains information about availability of free slots at the given planning cycle. The main objective of group scheduler is to allocate available free slots to processing elements which request for them and sharing of processing capacity within the group only. This helps us to decrease communication delay among processing elements. The next section proposes how to make use of these unused processing capacities for newly created dynamic tasks.

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 is sufficient to accommodate newly arrived task then that task is allocated to that particular processor otherwise task is split into two fragments such that one fragment is allocated to one of the available processors and other is allocated to another processor. The splitting can be performed in such a way that the task executes in given group only, otherwise this task is rejected.

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 is assigned to another processor m_j it reserves after time $t + x [m_i]$.

Assume S_1, S_2,…,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}.

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 the mentioned interval.

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

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

Where e_{ij} is the execution time of task i on processor j.

Likewise a bigger task can be split into smaller fragments and allocated to the processing elements.

When a dynamic task arrives at time t, the task tries to schedule locally otherwise it searches for free slots in that particular processor group. The algorithm 2 is used for getting number of free slots in each processor and to find the size of each free slot**.** The size_free_slots [req_size] and no_free_slots [req_size] give us the size of free slot and number of free slots in that particular interval for each and every processing element.

The Algorithm 2 is used for finding optimal free slots and their sizes for newly arrived tasks and the algorithm tries optimizing among two processing elements. This means the sum of execution units between two processors is just equal to the computation time of the task. The Algorithm 3 is used for finding free slots of maximum size within available free slots in each processing element. The Algorithm 2 and Algorithm 3 can be invoked when a new task arrives and the status of the availability of slots in the given interval should be known. If the slots are not available within the group, task is simply rejected and a log file is created for those rejected tasks. This helps us to increase or decrease the size of the group and will also be useful for proper assessment of task allocation.

Once a group scheduler identifies processor for required time units of a given task, it immediately locks the identified slots.

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

Algorithm 2. Finding optimal free-slot

```
//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]
       }
```

Finding maximum size of free-slot t and t+d

Algorithm 3. Finding maximum size of free-slot t and t+d

```
      Max(availability_fslot[])
      {
        Maximum value=0;
        for (j=0;j<req_size;j++)
        {
          If(locked[j]!=1)
          {
            If (Maximum value<availability_fslot[j])
               Maximum value= availability_fslot[j];
           }
        }
         return j, maximum value;
      }
```

The workload of given application is distributed among m processor system and each processor maintains local scheduler. The splitting of task can be possible within a certain set of processing elements and each set maintains group scheduler. The next section presents an integrated procedure to schedule different types of tasks.

5. 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 it is known that, there will be certain free slots available and these intervals are fixed.

Each group maintains Maekawa's sets [10] which are chosen in such a way that there will be a common communicating processing element among different sets that gives equal responsibility to every processing element in state information exchange, using a minimal number of messages. Each group of processing elements maintains the following three sets for efficient communication to other processing elements.

Request Set (R_i): Set of processing element to which the group sends request for state information.

Inform Set (I_i): Set of processing elements to which it sends information about its state.

Status Set (S_i): Set of processing elements in which free slots are available.

The group scheduler maintains a status table which contains Processor ID, fixed total time execution units already committed to run and number of free-slots in a given planning cycle. The fixed total execution units are statically allocated using partitioned algorithm known as RMCT and these tasks are known as critical tasks. The centralized scheduler distinguishes processing elements into groups of request sets considering certain parameters such as availability of free slots and communicating processing elements. For a given group of processing elements, the size of the set can be figured out which can be called set size. But number of processing elements in each group can be fixed and required number of groups can be computed. Each group can be given a unique id and there is no communication among the groups but processing elements share unused space within the group only. Set-size can be given by the square root of number of processing elements in a group. Where gn indicates that number of elements in a group.

$$\begin{aligned} &1.\forall_i \forall_j \left[R_i \cap R_j \neq \phi \right] \\ &2.\forall_i \left[m_i \in R_i \right] \\ &3.\forall_i \left[\left| R_i \right| = set\_size \right] \end{aligned} \qquad \ldots\ldots\ldots(6)$$

Processing element m_i is contained in exact set size request sets

There fore

$$set\_size = \sqrt{gn}$$

1. There is no common element between two groups such as i,j

$$G_i \cap G_j = \phi$$

$$\forall_i \forall_j : 1 = i, j = N :: G_i \cap G_j = \phi$$

The number of groups can be computed which depends on the number of processing elements and size of each group can be fixed. The total number of groups in a system is equal to ratio between processing elements (m) and size of the group (gn). REQUEST (i) is sent to all processing elements in each group to know the availability of free slots in that particular interval of time. All processing elements send RESPONSE (j) about availability of free-slots and its size.

Example 4.2: The processing elements m=100 divides into 15 groups and each group contains 7 processing elements. set_size=$\sqrt{7} \approx 3$. There three request set R_i (1, 2, 3), (3, 4, 5) (5.6, 7) are formed within the group. Each set has a processing element which communicates with the other sets in the group scheduler about the status of availability of free slots.

In order to use our algorithm, it is necessary to ensure that each task is processed on one processor only at any point of time. Task splitting must therefore address three important challenges.

1. Dispatching algorithm to be developed for ensuring that smaller pieces of a task do not execute simultaneously
2. Design a schedulable test for the dispatching algorithm.
3. Order of execution of all generated pieces is maintained properly.

The next section presents Schedulable Analysis.

5.1 Schedulable Analysis

The processor demand represents the amount of execution that must necessarily be given to a task in order to meet its deadline. The execution demand of newly arrived tasks at phase $Ø_i$, periodicity (p_i) and deadline d_i should always in between $Ø_i$ and $Ø_i+d_i$. The processor demand of $task_i$ is in the planning cycle (0, M] and all tasks which are released in this planning cycle should obey following conditions.

a) Should arrive not earlier than beginning of interval
b) Should have deadline no later than the finishing of time interval.

The processor demand of a task set is defined analogously. Let *dbf* (T_{total} , *M)* denote the processor demand of task set τ of time interval *(0, M]* where *M* is planning cycle. T_{total} is equal to sum of execution units of k tasks on given processor j.

$$T_{total} = T_j + T_{edf} \quad \text{..... (7)}$$

T_j= Fixed execution units allocated using theorem -1

T_{edf} = Addional execution units added to processor i.

The T_{edf} can be computed dynamically using earliest deadline first algorithm. When an aperiodic task arrives at phase Ø, execution time e_i and deadline d_i on processor m_i then invoke algorithm for finding free slots in given interval. S_{opt} be optimal execution units in a given processor group.

The space reserved for one portion of task should be equal to S_{max}

So, $x [m_j] = S_{opt}$

If the S_{opt} is equal to execution units e_i of task i then that task is allocated to the processor j otherwise suitable processor is searched in the same processor group. The remaining portion of split task can be computed as $Z[m_{j+1}]$.

i.e. $Z[m_{j+1}] = e_i - S_{opt}$

The T_{edf} for processor j and j+1 can be updated with $x [m_i]$ and $Z[m_{j+1}]$ respectively in the interval Ø and Ø +d.

T_j fixed units are allocated to individual processor using partitioned algorithm known as RMCT and T_{edf} is computed using free slots algorithms. The allocation of all newly arrived tasks in τ are met if and only if

$$\forall M > 0, dbf(\tau, M) \leq M \quad \ldots\ldots\ldots (8)$$

The above schedulable condition gives us processor demand of a $task_i$ in the of interval between task release time and its completion time. This means that total execution units are less than or equal to M. The above condition fails when there is no enough free space on the processor to accommodate a newly arrived task. So in this scenario the task should be split according to algorithm mentioned above.

6. PERFORMANCE RESULTS

By making some minor changes to the set up used by Anderson [4], the periodicity (p) values ranging from 1 to 1000 are uniformly distributed. Experiments are performed for two processor and eight processor systems. In our simulation model, execution times may be drawn only from bimodal. According to Bimodal tasks can be heavy and light which can be given any random utilization value from the intervals [0.5, 1) and (0.0, 0.5) respectively. The execution times of task τ_i are chosen such that it is equal to p_i times randomly chosen utilization value in given interval. Experiments were performed for periodic task sets and deadline of each task should be equal to its periodicity.

Initially using partitioned scheduling algorithm, fixed number of execution units are allocated to the individual processor with fixed value of configuration parameter (k). The task set generator which is developed is used to check feasibility of individual task set and individual task. The task sets with $\frac{1}{m}\sum_{i=1}^{n}\frac{e_i}{p_i} \geq 1$ are immediately rejected by the task set generator and also at least one task in task set which has $e_i > p_i$ is rejected as well.

We run multiple experiments and in each experiment task set generator developed by us is used which generates more than 1000 task sets. These task sets are written in a file and this file is fed as input to our algorithms. For simulation J.Anderson and RMCT_Split algorithm are implemented and the same input is given to these two algorithms.

For each task set, the utilization is calculated and it is put it into the bucket. There are 100 buckets and each bucket is used for each percentage of utilization. For example all task sets with utilization within [0.5 0.4) are put the one bucket. For each bucket the number of task sets can be calculated which are scheduled with each algorithm considered. All tasks in a given task set meet feasibility conditions otherwise particular task set is rejected by our generator. Our simulation results can however be used to find number of task sets feasible for m=2 and m=8 as well. This gives us a few tasks which can be split and allocated to two processor or eight processor systems. The results from bimodal distribution are shown in Figure 5 to 6.

The y-axis should have been labelled "number of task sets "and x-axis "total utilization of each task set". The graph tells us the number of task sets that were schedulable for each utilization range. In turn, each task set might consists of 10, 18, 25…etc. tasks or even small number of tasks but we are interested about total utilization of each task set. From these experiments we conclude that more number of free slots is available in RMCT algorithm and these free slots are utilized for newly created aperiodic tasks. By adjusting parameter k more number of aperiodic task sets is allocated to a processor. However, processor utilization never exceeds one but we should have proper mix of aperiodic tasks and periodic tasks.

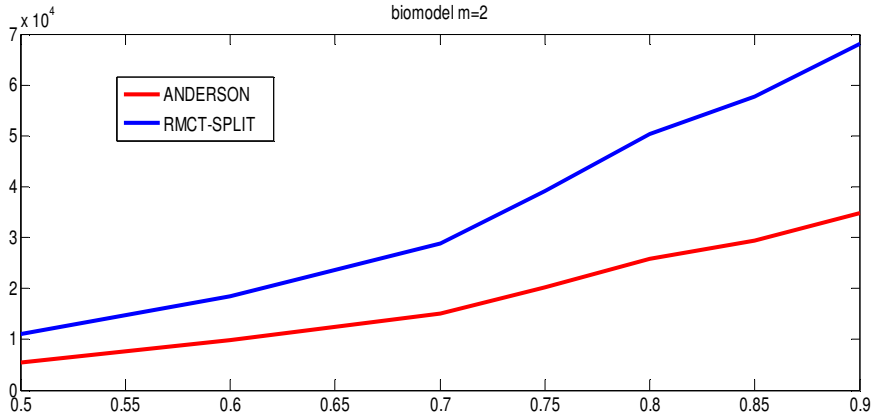

Figure 5. Bio Model m=2

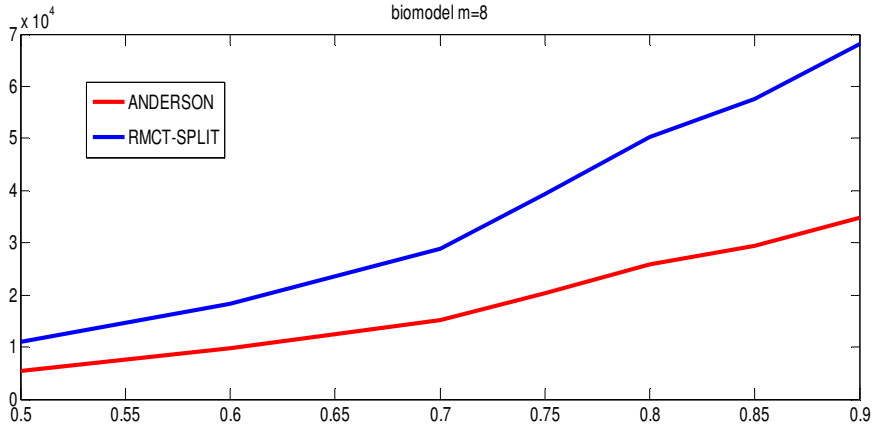

Figure 6. Bio Model m=8

6. CONCLUSIONS

This paper provides solution to make use of unused free-slots and to schedule different types of tasks with periodic task sets. Initially, the newly created dynamic tasks were tried to schedule to one of the processors available but in a few cases CPU cycles were not sufficient to execute a given task. In such scenarios splitting can take place between two processors which improves better resource utilization. Only a few processors are grouped together using Mekava set which reduces communication delay. Simulations were conducted for 100 to 200 tasks. This work which includes RMCT algorithm to schedule different types of tasks shows better results than that of the previous work. Whatever algorithms which were developed are purely deterministic and task nature was predictable in that particular type of environment.

This paper predominantly considers a control strategy for allocating different types of tasks or activities are scheduled with periodic tasks. Also strategy for allocating at least on periodic task to an individual processing element and remaining unused cpu cycles are used for allocating different activities. The same principle can be extended to schedule different activities in shopping mall and traffic monitoring system. In shopping mall and traffic monitoring system communication among customer through mobile devices. In this scenario all customer activities are monitored through mobile devices and each device possesses at least one sensor and rests of CPU cycles are used for other activities. Our feature work performs real-time analysis of sensed data to collaborate to increase computation power and development such type of device.

REFERENCES

[1] C.Liu and J.LayLand, "Scheduling Algorithms for Multiprogramming in Hard Real–Time Environment," JACM 10(1), 1973, pp, 174-189

[2] A.Prashanth Rao and A. Govardhan, "An Improved Period Oriented Scheduling Algorithm for Real Time Systems." Ijesce Research Science Press, Vol 3, No.1 January-June 2011.

[3] S.Cheng, J.A.Stankovic, and K.Ramamritham, Scheduling Algorithm for Hard Real Time Systems: A Brief Survey, Tutorial: Hard Real Time Systems, EFF Press, 1988, pp 150-173.

[4] J. D Gafford, Rate Monotonic Scheduling, IEE Micro, June 1991, pp 34-39 Real Time Systems by James W.S.Liu Published by Pearson Education, II Ededition, 1991.

[5] D.S Johnson, Near Optimal Bin Packing Algorithms, PhD Thesis, MIT 1973.

[6] Giorgio C.Buttazzo, Hard Real-Time Computing Systems Predictable Scheduling Algorithms and Applications, Kluwer Academic Publishers,1997

[7] Edited by Mathai Joseph, Real-time Systems Specification, Verification and Analysis, Tata Research Development & Design Centre, June 2001

[8] Almut Burchard, Jorge Liebeherr, Member, Yingfeng Oh, and Sang H. Son, New Strategies for Assigning Real-Time Tasks to Multimocessor Systems, IEEE Transactions On Computers, Vol.44, No.12, December 1995

[9] Sylvain Lauzac, Rami Melhem, Fellow, IEEE, and Daniel Mosse´, Member, IEEE Computer Society An Improved Rate-Monotonic Admission Control and Its Applications, IEEE Transactions On Computers, Vol. 52, No. 3, March 2003

[10] C.Siva Ram Murthy and G.Manimaran, Resource Management in Real-Time Systems and Networks, PHI Learning Private Limited, 2009.

[11] D. Zmaranda, G. Gabor, D.E. Popescu, C. Vancea, F. Vancea, Using Fixed Priority Pre-emptive Scheduling in Real-Time Systems, International journal of Computers, Communications & Control, Vol.VI (2011), No.1(Marach), pp.187-195.

[12] Rodolfo Pellizzoni,Giuseppe Lipari,Feasibility Analysis of Real-Time Periodic Tasks with Offsets, Real-Time Systems,Kluwer Academic Publishers Norwell, Volume 30 Issue 1-2, May 2005

[13] Bjorn Anderson, Konstantin's Bletsis, Sanjoy Baruah, Arbitrary-Deadline Scheduling Sporadic Tasks on Multiprocessor, Technical Report HURRAY-TR-080501.