Slot Shifting

L J GOKUL VASAN

INTRODUCTION

- An algorithm that tries to give schedulable guarantee for Event Based tasks within the periodic task-set that is offline guaranteed.
- Event Based Tasks can be classified as
  – Soft aperiodics
    - tasks with no deadline
  – Firm aperiodics
    - tasks with deadlines but dl miss will not collapse system
  – Hard aperiodics
    - Task with deadlines but dl miss will collapse/cause fatal system failure.
- Slot shifting can only support Soft and Firm aperiodics.

Overview on Working

- The Algorithm Works in 2 phases
  1. Offline Phase
  2. Online Phase
- Fundamentals of online Selection Function is EDF.
- The Decision Function is based on uniform time intervals called **SLOTS.**

1. OFFLINE PHASE

- Basic schedulable test is done on **periodics** .
- **A layer of certainty** is applied on periodics called **Interval** notated as **I** defined with following terms**.**
  - End(I) *: deadline(J_i). i $\in$ all jobs with same deadline*
  - Est(I) *: min (est(J_i)). i $\in$ all jobs with same deadline*
  - Start(I): *max(est(I_i), end(I_{i-1}))*
  - Length(I): *| I_i | $\rightarrow$ End(I_i) – Start(I_i).*
  - *SC(I) : | I_i | - ∑WCET(J_i) + min(SC(I_{i+1}),0) , Spare Capacity of the interval I*
  - So Basic record of interval looks like:

```
struct interval {
    int id; /// unique ID for the interval
    int start; /// start of interval
    int end; ///end of interval
    int sc; /// spare capacity of interval
    list *jobs; /// list of jobs that belong to this interval
    list *nxt; /// reference to next interval / NULL
};
```

1. OFFLINE PHASE

```
function create_new_interval()
        input s, e: s is the start of interval
                    e is the end of interval
        output i:  i is the created new interval

        i := allocate memory for new record of type struct interval
        i.start := s;
        i.end := e;
        i.jobs := NULL;
        i.nxt := NULL;
        return i;
-----------------------------------------------------------------------------------------
function get_first_job_dl()
        input j: j is list of jobs
        output j⁰:  j⁰ is the 1st job in the list / NULL.
        if(j) return j[0];  else return NULL;
-----------------------------------------------------------------------------------------
function get_rmv_1st_job()
        input j: j is list of jobs
        output j⁰ : j⁰ is the 1st job in list , removed from list / NULL
        if(j)
            j⁰ = get_first_job_dl(j);
            remove_job(j, j⁰ );  /// removes job j0 from list j
            return j⁰ ;
        else  return NULL;
-----------------------------------------------------------------------------------------
```

```
function CreateInterval()
        input  T: T is schedulable periodic task-set
        output t,I: t is list of jobs sorted based on dl.
                    I is list of intervals
        st := sort jobs of task-set based on deadline.
        t := st  ///duplicate list of st on t
        while(st)  //CREATE INTERVAL
            I.nxt = create_new_interval(0,0);
            I.end = get_first_job_dl(st);
            while(I.end == get_first_job_dl(st))
                I.jobs.nxt =   Get_rmve_1st_job(st);
            est_i = min(I.jobs.est);
            I.start = max( est_i, I_{i-1}.end);
        while(I)  //CREATE SC OF INTERVAL
            I.sc = |I| - ∑WCET(I.jobs) + min(I_{i+1} .sc, 0);
            I = I.nxt;
        return t,I;
```

2. ONLINE PHASE

- Basic selection function is based on EDF on Guaranteed ready list or FCFS on !Guaranteed list.
- Selection function also manages updating intervals and its spare capacity.
- It checks Firm aperiodics arrival; if the presence is detected then acceptance test is run and if it can be guaranteed then guarantee algorithm is run.
- If Soft aperiodics arrives, then simply add the task to not-guaranteed list and schedule them when slot is empty after guaranteed EDF function.

2. ONLINE PHASE

```
Selection_function ( )
input t_prev : t_prev is the previous task that was scheduled.
output t_nxt : t_nxt is the next task to be scheduled.

  /// 1.INTERVAL UPDATE
    update_sc (t_prev); /// update spare capacity of the current interval based on prev scheduled task.
    I = Get_current_interval();
   if(slot_count > I.end) /// check we are on end of interval, if so move to next interval
       move_to_next_interval();
   end if

   /// 2.APERIODIC CHECK
   If(aperiodic_task)  ///check on aperiodics task
        if(FIRM  == aperiodic_task`.type)
              acceptance_guarantee_algorithm(aperiodic_task); /// run acceptance and guarantee algorithm
        else if(SOFT  == aperiodic_task.type )
            add_to_notGuaranteed_list(aperiodic_task);
`         end if

   ///3.EDF
    t_nxt = get_nxt_guaranteed_ready_task();  /// EDF  on ready list
    if(! t_nxt )
         t_nxt = get_nxt_notGuaranteed_ready_task();  ///FCFS on not guaranteed ready list
    end if
    return  t_nxt ;
```

2. ONLINE PHASE

```
function Update_till_positive()
Input I : I is the interval from where sc needs update in
        backwards.

  i_tmp = I;
  do{

      i_tmp.sc++;
      i_tmp = get_prev_interval(i_tmp);
     if(i_tmp == get_current_interval())
          break;
     end if

  }while(i_tmp.sc < 0);

  i_tmp.sc++;
```

```
function Update_sc()
input t: t is the job that got scheduled

    I := get_current_interval();
    while(I.job)
        if(I.job == t) return;
        I.job = I.job.nxt;

    i_tmp = get_task_interval(t);
    if(i_tmp .sc  < 0)
       update_till_positive(i_tmp);
    end if

    negate_sc(I);
```

2. ONLINE PHASE

FIRM APERIODIC ARRIVAL (J_A): *acceptance_guarantee_algorithm*(J_A)

- On Firm aperiodic arrival acceptance test needs to run on task to check whether it can be admitted.
- To run the **acceptance test** we traverse through 3 kinds intervals and sum there SC.
  1. $SC(I_C)$: Spare capacity of current interval.
  2. $SC(I_i)$. $C < i <= l$. $end(I_l) <= dl(J_A) \wedge end(I_{l+1}) > dl(J_A)$.
     - All intervals that are between current and last interval.
  3. $Min[SC(I_{l+1}), dl(J_A) – start(I_{l+1})]$
     - Required spare capacity in the last interval within $dl(J_A)$.

  If(1+2 +3 >= WCET(J_A)) then J_A can be accepted i.e. return true.
- Simple snippet of acceptance guarantee Algorithm :

```
Acceptance_guarantee_algorithm(J_A)
    if(acceptance_test(J_A))
        guarantee_algorithm(J_A);
    else
        move_to_notGuaranteed_list(J_A);
```

2. ONLINE PHASE

TRADITIONAL GUARANTEE ALGORITHM:

Function Guarantee_algorithm(J_A)

$if(dl(J_A) < end(I_{l+1}))$

$split\_interval(I_{l+1}, J_A);$

$I_i = I_{l+1};$

$while(I_i \mathrel{!=} I_c)$

$sc(I_i) = |I_i| - \sum WCET(J_i) + min(SC(I_{i+1}),0);$

$I_i = get\_prev\_interval(I_i);$

Complexity : **O(N.t)** where N is the number of intervals and t is the jobs in that intervals.

- **Redoing what we did offline and traversing both vertically and horizontally in a list is very cumbersome to implement.**
- **We tried to use this offline effort to recalculate online change in SC without having vertical traversal. Giving us O(N) guarantee algorithm.**

2. ONLINE PHASE : *O(N) GUARANTEE ALGORITHM*

```
Function split_interval ( )
Input : I_right , split_point , curr_slot
Output : I_left

If(I_right == get_current_interval() )
        new_len = (split_point  + start(I_right)) – curr_slot;
else
        new_len = split_point + start(I_right) – start(I_right);

SC(I_right) = SC(I_right) – new_len;

SC(I_left) = new_len + min(0, SC(I_right));
END(I_left) = split_point + start(I_right);
START(I_left) =  start(I_right);

START(I_right) = split_point + start(I_right);

Add_intr_to_list (I_left);
return I_left
```

Function **Guarantee_algorithm**( )

Input I_{dl} , $J_{A,}$, slot_no : **I_{dl} is the interval in which aperiodic deadline ,i.e. I_{l+1}**
J_A is the aperiodic task that cleared acceptance test.
slot_no is the current slot number.

Output : NONE

```
If(dl(J_A,) <  END(I_dl,))
        split_point  = DL(J_A,)  - START(I_dl);
        I_left  = split_interval (I_dl,  split_point , slot_no);

J_A . Interval  = I_left  ? I_left  : I_dl  ;
delta  := WCET(J_A ) ;
i_temp  :=  J_A .Interval;

while(delta)
        if(SC(i_temp) > 0)
                if(SC(i_temp) >= delta)
                        SC(i_temp)  = SC(i_temp)  -  delta;
                        delta  = 0;
                else
                        delta = delta – SC(i_temp);
                        SC(i_temp) = -delta;
        else
                SC(i_temp) += -delta;

        i_temp  = PREV(i_temp);
```