LRE-TL: An Optimal Multiprocessor Scheduling Algorithm for Sporadic Task Sets

Shelby Funk
Vijaykant Nadadur
Multiprocessor real-time systems

• Increasingly, real-time systems are developed on multiprocessors
  - Many popular uniprocessor real-time scheduling algorithms do not perform well on multiprocessors

• Optimal multiprocessor real-time scheduling algorithms suffer from one of two shortcomings
  - They impose high system overhead
  - They apply to restricted processing models
Task model

- **Periodic & sporadic tasks:** A mechanism for executing jobs repeatedly
- \( T_i = (\phi_i, p_i, e_i) \)
  - \( \phi_i = \) offset
    - Periodic task: release time of first job
    - Sporadic task: earliest release time of first job
  - \( p_i = \) period
    - Periodic task: Exact time between jobs
    - Sporadic task: Minimum time between jobs
  - \( e_i = \) worst-case execution time
- **If a task** \( T_i \) **generates a job at time** \( a_i \), **then it must be allowed to execute for** \( e_i \) **time units during the interval** \( [a_i, a_i + p_i] \)
Task utilization

• Given a periodic or sporadic task \( T_i=(p_i,e_i) \), the utilization of \( T_i \) is \( e_i/p_i \)
  - Average proportion of processing time this task will require

• Given a set of tasks \( \tau = \{T_1, T_2, \ldots, T_n\} \)
  - \( U(\tau) = \tau 's \) total utilization = \( \Sigma (e_i/p_i) \)
  - \( u_{max} = \tau 's \) maximum utilization = \( \max\{e_i/p_i\} \)

• Many tests are based on task utilization
  - \( \tau \) can be scheduled on an m-processor identical multiprocessor iff \( U(\tau) \leq m \) and \( u_{max} \leq 1 \)
Online multiprocessor scheduling

• The utilization test states *some* schedule exists
  - This schedule might be difficult to implement

• Hong and Leung proved that no online scheduling algorithm can be optimal when deadlines are not all equal
Ideally, we would execute all tasks at a constant rate.

- Example $T_1 = (2,4)$, $T_2 = (3,5)$, and $T_3 = (6,8)$
LLREF overview

• The timeline is broken into TL planes
  - Time and Local execution time
  - Dividing points are determined by task deadlines
  - Scheduling within a TL plane $[t_{i-1}, t_i]$ ensures tasks have executed at their ideal amount at by time $t_i$

• Example $T_1 = (4,2), T_2 = (5,3), \text{ and } T_3 = (8,6)$
LLREF schedules

• Tasks are guaranteed to have executed at ideal rate on every deadline
  - Hence, all deadlines are met - LLREF is optimal for multiprocessors
Scheduling TL planes

- Each task is assigned a local workload for the given TL plane \([t_e, t_f]\)
- \(\ell_i = \) remaining time for \(T_i\) in current TL plane
  - At start of each TL plane \(\ell_i = u_i \cdot (t_f - t_e)\)
- \(r_i = \) local utilization within TL plane
  - \(r_i = \ell_i / (t_f - t_{cur})\)
- \(R_t = \) total local utilization at time \(t\)
- The \(m\) tasks with largest local remaining execution time are selected to execute
  - Tasks are scheduled until a B or a C event
Scheduling events

- Within the plane, each task is represented with a token
  - A correct schedule keeps all tokens within the plane
- A new scheduling event occurs under two conditions
  - B (bottom) events: A task’s token hits the bottom of the TL plane
  - C (critical) events: A task’s token hits the NLLD
LLREF scheduler

- At every scheduling event, the m tasks with the largest local execution execute until a B or a C event occurs
  - LLREF = Largest Local Remaining Execution First
Example

$T_1 = (7,2)$, $T_2 = (10,3)$, $T_3 = (9,4)$, $T_4 = (12,7)$, $T_5 = (14,5)$
LLREF complexity

- When a B or C event occurs the following steps are taken
  - Remaining execution time of m executing tasks are updated $O(m)$
  - Tasks are sorted by local remaining execution time $O(n)$
  - The m tasks with highest remaining execution time are selected to execute $O(1)$
  - The earliest upcoming B and C events are determined $O(1)$
  - Scheduling proceeds until that event
Reducing overhead

- Recall Hong and Leung’s claim: No online scheduling algorithm can be optimal when deadlines are not all equal
  - When deadlines are equal, there only two concerns
    - Ensuring jobs with zero laxity execute immediately
    - Ensuring jobs with zero remaining execution do not execute

- LLREF makes all deadlines equal within a TL plane
- We can remove the sort step from the scheduling algorithm
Theorem

- Assume that \( \tau \) is executing on \( m \) processors and the following conditions hold at time \( t \):
  - The total local utilization of \( \tau \) is at most \( m \)
  - The maximum local utilization of \( \tau \) is at most 1
  - There are \( m \) tasks with non-zero remaining local execution time

- If any \( m \) tasks execute and no task’s local execution time becomes negative, then local utilization will not increase over time
Modified task maintenance

• Two min heaps: B-heap and C-heap
  - key = Amount of time to a B/C event
• When an event occurs, preempt as few tasks as possible
  - No preemption for B events - simply select any non-executing task to execute
  - Preempt one task for C event - schedule the zero laxity task on the freed processor
New scheduling event

• The theorem observed that total local utilization does not decrease over time

• If a sporadic task generates a job within a TL plane, we invoke an A event
  - Let $\ell_i = u_i \cdot (t_f - t_{cur})$, where $t_f$ = end of TL plane and $t_{cur}$ = arrival time
  - Add $T_i$ to the non-executing heap
    • If $u_i = 1$, $T_i$ must preempt some executing task

• Total local utilization at A event is at most $m - u_i$ and $T_i$’s local utilization = $u_i$
  - Total utilization cannot exceed $m
Two schedules

- Compare the schedules for $T_1 = (7,2)$, $T_2 = (10,3)$, $T_3 = (9,4)$, $T_4 = (12,7)$, $T_5 = (14,5)$
Algorithm comparison

<table>
<thead>
<tr>
<th></th>
<th>LLREF</th>
<th>LRE-TL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of B/C events per TL plane</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Max number of preemptions during each event</td>
<td>O(m)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Running time per B/C event</td>
<td>O(n)</td>
<td>O(log m + log (n-m))</td>
</tr>
<tr>
<td>Running time per A event</td>
<td>--</td>
<td>O(log (n-m))</td>
</tr>
</tbody>
</table>
Conclusion

- Introduced method to reduce complexity of LLREF
- Introduced method to allow for sporadic task arrivals
- In future, we hope to do the following
  - Reduce complexity even further
  - Apply results to uniform multiprocessors
  - Incorporate resource sharing
  - Apply to DVFS multiprocessing
A-event special case

• What if $T_i$ invokes a job with deadline $d_i$, where $d_i < t_j$?
  - Deadline within a TL-plane is not allowed!

• Split remainder of TL-plane into two pieces
  - $[t, d_i)$ and $[d_i, t_j)$

• $T_i$ executes for $e_i$ time units during $[t, d_i)$

• Any other task $T_j$ has $l_{j,\dagger}$ divided into two pieces
  - Size of pieces proportional to length of the two intervals