MC Scheduling on Varying-Speed Processors

Zhishan Guo
University of North Carolina at Chapel Hill
zsguo@cs.unc.edu

Keywords: varying-speed processors, model combination

1 Introduction and Motivation

Most existing research on Mixed-Criticality (MC) scheduling (see [1] for a review) has focused on dealing with different WCET estimations of a single piece of code. This is typically a consequence of different tools for determining worst case execution time (WCET) bounds being more or less conservative than each other.

This narrative is now being repeated with respect to processor speeds. Modern powerful and energy-efficient processors are yielding innovations that result in varying speed during run-time. For example, [2] describes a mechanism such that late signals can be recovered by delaying the next clock tick, so that logical faults do not propagate to higher (i.e., the software) levels. In a Globally Asynchronous Locally Synchronous (GALS) circuit, local clocks can be affected by signals propagating between different synchronous modules in an asynchronous manner.

Research on such varying-speed platform may lead to better understanding of a wider range of problems. For example, in data communication of automobiles, aircrafts, or wireless sensor networks, time-sensitive data-streams must be transmitted over potentially faulty communication channels, where a high bandwidth is provided under most circumstances yet only guaranteeing a lower bandwidth.

2 Model and Existing work

A varying-speed processor is modeled as follows: under normal circumstances, it completes at least one unit of execution during each time unit, while it may fall into a degrade mode at any instant, during which it can only complete \( x \in [s, 1) \) units of execution during each time unit, for some (known) threshold \( s < 1 \). It is not a priori known when, or whether, such degradation will occur. Similar to other MC scheduling problems, we seek a strategy that guarantees to complete all jobs by their deadlines under normal (LO-criticality) behaviors, while simultaneously guaranteeing to complete all HI-criticality jobs if either the platform (or the jobs) suffer from degradation (HI-criticality) behaviors. Note that here we are considering a combination of various aspects that MC may arise from, including periods, WCETs, processing speeds, etc.

Based upon the properties of the platform and the workload, we classify those problems into four categories:
Table 1: Existing work on scheduling MC sets on varying-speed uniprocessor.

<table>
<thead>
<tr>
<th></th>
<th>Jobs &amp; S-WCET</th>
<th>Tasks &amp; S-WCET</th>
<th>Jobs &amp; M-WCET</th>
<th>Tasks &amp; M-WCET</th>
</tr>
</thead>
<tbody>
<tr>
<td>NM Uniproc.</td>
<td>[8]</td>
<td>[3]</td>
<td>[6]</td>
<td>[3]</td>
</tr>
<tr>
<td>SM Multiproc.</td>
<td>[7]</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NM Multiproc.</td>
<td>-</td>
<td>-⁴</td>
<td>-</td>
<td>-⁴</td>
</tr>
</tbody>
</table>

(i) Self-Monitoring: A self-monitoring (SM) processor immediately knows its execution speed during run-time¹, while non-monitored (NM) one may not.
(ii) Number of processors: Either uniprocessor, or multiprocessor.
(iii) Workload model: One shot job set, or sporadic/periodic task set.
(iv) Single(S)- or Multiple(M)- Worst case execution time (WCET) per job.

Table 1 lists existing work on MC scheduling upon varying-speed platforms.

3 Further Directions

Most current work only deals with one-shot jobs or implicit-deadline sporadic tasks, and the generalization to constrained deadlines is not trivial. Also, as shown in Table 1, much remains to be done regarding multiprocessors – the degraded mode upon such platforms needs to be completely specified. If different processors are assumed to degrade to different speeds, the resulting degraded platform may become a heterogeneous one, for which the MC scheduling problem is totally open.

References


¹Similar to Linux command cpufreq-info, SM platform has access to processor speeds, while NM processor may only identify degradation upon some job not signaling its finishing on time.
² The strong NP-hardness of non-preemption scheduling under such case is also shown in [4] and [5].
³ Regarding scheduling tasks, [4] and [5] only provide necessary conditions and a corresponding optimal sharing-based (fluid) scheduling scheme, which is impractical due to too many preemptions.
⁴ We may model a NM varying-speed processor with the multi-WCET MC model, and apply some existing MC scheduling work, e.g., [9], while being somewhat pessimistic, which is similar as [3].